Doctor of Philosophy

Business Process Discovery Algorithms Recommendation Framework and Heuristic Rule−based Process Discovery Algorithm Development

The Graduate School
of the University of Ulsan

Department of Industrial
Engineering
Hind R’ bigui

Business Process Discovery Algorithms Recommendation Framework and Heuristic Rule−based Process Discovery Algorithm Development

Supervisor : Prof. Chiwoon Cho

A Dissertation

Submitted to
the Graduate School of the University
of Ulsan
In partial Fulfillment of the
Requirements
for the Degree of

Doctor of Philosophy

by

Hind R’ bigui

Department of Industrial Engineering
University of Ulsan, Ulsan, Korea
December 2018

Business Process Discovery Algorithms Recommendation Framework and Heuristic Rule-based Process Discovery Algorithm Development

Jae Gyun Kim [illegible]

Committee Chair Dr.

Chiehyeon Lim [illegible]

Committee Member Dr.

Myung Bock Kong [illegible]

Committee Member Dr.

Kihyo Jung [illegible]

Committee Member Dr.

Chiwoon Cho [illegible]

Committee Member Dr.

Department of Industrial Engineering

Ulsan, Korea

December 2018

ABSTRACT

Under pressure of the rapid change engendered by the fast growth of information and communication technologies, organizations need to continuously enhance their business processes in order to defend their market position and maintain their competitive edge. To achieve this, process mining has emerged as a mean to analyze the behavior of companies. Business process mining is new methods that amalgamate business process modeling and analysis with data mining, artificial intelligence and machine learning techniques, whereby process-oriented knowledge from event logs stored in today′s information systems are extracted to automatically discover business process models, identify bottlenecks, and improve the business processes. Many powerful process discovery algorithms have recently been developed. However, users and businesses still cannot choose or decide the appropriate mining algorithm for their business processes. Each algorithm has a specific limitation regarding the mining of short loops, invisible tasks, duplicate tasks and non-free choice constructs. There is no algorithm which is capable of discovering the aforementioned characteristics in a restricted time if all of them are present in the event log.

The goal of this research consists of first developing a process discovery algorithms recommendation framework capable of recommending to businesses the most suitable process discovery technique to their business processes based on the knowledge in the event logs of the processes in question; second developing a new process discovery algorithm capable of handling standard constructs, short loops, invisible tasks, duplicate tasks, and non-free choice constructs if all of them exist in the event log.

초록

정보통신기술의 급속한 성장으로 인한 급속한 변화의 압력을 받고, 기업은 시장 지위를 방어하고 경쟁 우위를 유지하기 위해 비즈니스 프로세스를 지속적으로 향상시켜야 한다. 이를 달성하기 위해, 프로세스 마이닝은 기존 정보 시스템에 기록된 이벤트 로그에서 프로세스 지향 지식을 추출하여 기업의 행동을 분석하는 수단으로 떠올랐다.

프로세스 마이닝은 비교적 새로운 연구 분야로, 인공 지능 및 데이터 마이닝과 프로세스 모델링 및 분석의 중간에 위치한 연구 분야다. 프로세스 마이닝의 기본적인 아이디어는 정보시스템에 기록되어 있는 이벤트 로그에서 지식을 추 출함으로써, 사람들이 머리 속에서 추정하고 있는 프로세스가 아닌 실제 업무 프로세스를 자동으로 도출하고, 모니터링하며, 개선하는 것이다. 이벤트 로그는 세 가지 유형의 프로세스 마이닝 수행에 이용될 수 있다. 프로세스 마이닝의 유형은 도출, 적합성, 또 향상이다. 프로세스 모델 도출은 가장 중요한 프로세스 마이닝 기법이다.

최근에는 많은 강력한 프로세스 모델 도출 알고리즘이 개발되었다. 그러나 사용자와 비즈니스는 여전히 비즈니스 프로세스에 적절한 도출 알고리즘을 선택하거나 결정할 수 없다. 각 알고리즘은 짧은 루프, 보이지 않는 작업, 중복 작업 및 자유롭지 않은 선택 구조의 마이닝에 대한 특정 제한을 가지고 있기 때문이다. 이 모든 특징을 제한된 시간에 도출할 수 있는 알고리즘이 없다.

이 연구의 첫 번째 목표는 해당 프로세스의 이벤트 로그에 있는 지식을 기반으로 비즈니스 프로세스에 가장 적합한 프로세스 도출 기술을 비즈니스에 추천할 수 있는 프로세스 도출 알고리즘 추천 프레임 워크를 개발하는 것이다. 연구의 두 번째 목표는 표준 구조, 짧은 루프, 보이지 않는 작업, 중복 작업 및 비자유 선택 구조와 같은 프로세스 모델 특징을 처리할 수 있는 새로운 프로세스 도출 알고리즘을 개발하는 것이다.

ACKNOWLEDGEMENT

This thesis would not have existed without the help and support of many people and I would like to give my sincere gratitude in this humble acknowledgment.

First and foremost, I would like to express my special appreciation and utmost thanks to my supervisor, Prof. Dr. Chiwoon Cho who provided the inspiration and motivation required to make this thesis a success. I would like to thank him for his continuous support, for his consistent encouragement, for his patience, for his motivation, and for his immense knowledge that he provided me throughout this research. It was a great opportunity to do my doctoral programme under his guidance and to learn from his expertise. I would like to thank him also, for his continuous moral and financial support, for all his help, and for his advices on both research as well as on my career which have been priceless. Also, I would like to thank him for providing me a great opportunity to join the K-MOOC (Present and Future of Korean Industries) team as an assistant, to join a Korean government project, and to manager SAP users. I would like to express my sincere, for providing me real data from companies using his connections which played a huge role in the success of my thesis.

Besides my advisor, I would like to thank my thesis committee members: Professor Dr. Jae Gyun Kim, Professor Dr. Kihyo Jung, Professor Dr. Myung Bock Kong, and Professor Dr. Chiehyeon Lim, for serving as my committee members and for their valuable comments.

I would like to express my sincere gratitude to all the professors and people who have been accompanying me during my study at the University of Ulsan. I would not have been able to complete my research without their help. I would like to thank each and every one of them.

I would like to give my deepest gratitude to Dr. Mohammed Abdulhakim Ahmed Alabsi for his continuous encouragement throughout my doctoral study, his many helpful discussions to improve my thesis, for his great help in implementing my research work and for his consistent moral support and without him, I would not have accomplished nearly as much as I have. He deserves my deepest gratitude for his support.

I am extremely thankful to my best friend Dr. Safae Al Achhab for her consistent and continuous spiritual encouragement throughout my doctoral study and my life in general. I would not have accomplished nearly as much as I have without her being there to support me and encourage me.

I would like to give my special regards to Prof. Dr. Ahmed Al-Absi and my friend Dr. Assia Babikir Ahmed who were always willing to help, support and give the best suggestions that gave me the enthusiasm and courage to work hard. Special thanks go to Dr. Amir Tjolleng and Dr. Park Seonghyeon for their help and support and to all my friends without being mentioned one by one.

In addition, I would like to give my very profound gratitude to my beloved Grandmother Khadija Benzaid, my brother Younes R'bigui and his wife, my sister Soukaina R'bigui and all my family for being there to support and encourage me spiritually throughout writing this thesis. I thank them for all their loves and patience to come this far with me. You provide me the strength that I need to go forward.

My heartfelt gratitude goes to the Korean Government Scholarship Program and NIIED for giving me the opportunity of being a KGSP scholar which enabled me to carry out my doctoral degree throughout the past fourth years.

I would like to thank Allah, the Almighty, for blessing me and guiding me throughout my study in Doctoral degree.

Finally, the research described in this thesis is supported by the National IT Industry Promotion Agency (NIPA) and funded by the Korea government (MSIP) (S1106-16-1001, the development of the Shipyard IHOP process standard simulator and decision support system).

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

Chapter 1 INTRODUCTION

1.1. Process Aware Systems Limitations

Under pressure of the rapid change engendered by the fast growth of information and communication technologies, organizations need to continuously enhance their business processes in order to defend their market position and maintain their competitive edge. Most of the organizations adopt Business Process Improvement (BPI) approaches, Business Intelligence (BI) tools, and/or Business Process Management (BPM) techniques to enhance their operational performance and gain a competitive advantage in the common market. The main focus of Business Process Improvement (BPI) is to increase the satisfaction of the customer by enhancing quality and service, reducing expenses, and boosting the productivity of a business process or activity. Business process improvement creates an incremental enhancement in business processes (Zellner, 2011) via multiple existing management techniques such as Six-Sigma, Total Quality Management (TQM), Lean Management, Continuous Process Improvement (CPI), Agile Management, etc. Business Intelligence (BI) is a set of tools and techniques that use business related event data to measure the performance of a process or organization, support decision making, and enhance the business in real time. Unfortunately, the focus of BI tools is tailored toward data and local decision making rather than end-to-end processes (Van der Alast et al., 2012). Business Process Management (BPM) is considered one of the most promising disciplines for the organization as it allows the definition, monitoring, configuration, analysis and optimization of business processes (Zaouali et al., 2016). The aim of BPM is to provide the organization with end-to-end process understanding, visibility and control while ensuring efficient communication in an organization. In BPM systems, process models are used to analyze the “as-is” and “to-be” processes. Nevertheless, these models are absolutely disassociated from actual data as they are based on the idealized model of reality rather than real observations.

1.2. Process Mining

Process mining is a recent set of techniques which provides a strong bridge between BI and BPM by combining both process models and event data forming a novel form of process-driven analytics. Moreover, Process mining enables and strengthen the BPI approaches such TQM, Six-sigma, CPI, and so on, where processes are diagnosed to explore possible improvements.

Current information systems such as workflow management systems (WFM), enterprise resource planning (ERP) systems, customer relationship management (CRM) systems, supply chain management (SCM) systems, and business to business (B2B) systems store business-related events in so-called event logs (Van der Alast et al., 2007). These event logs usually contain information about the activities that have been executed in the enterprise (i.e., process instance), the time in which these activities were executed, the people, machines, or systems that handled those tasks and other data. Process mining extracts knowledge from these events logs to automatically build a representation of the current execution of business processes of an enterprise for the aim to identify incorrect executions, bottlenecks and other problems that prevent the organization from achieving its strategic goal and vision (Taylor et al., 2012).

Process mining uses these event data to extract process related information and three major classes of process mining techniques can be conducted for different purposes as illustrated in Figure 1.1: process discovery, conformance checking, and process enhancement or performance analysis (Van der Alast et al., 2011). Process discovery is considered the most important process mining technique. This type takes only an event log as input and automatically constructs a process model. Conformance checking compares an existing process model with the event log of the same process model to investigate that what is actually happening in the organization is in conformance with the process model. Process enhancement tends to enhance or extend the existing process model based on the information obtained from the discovered process model, or based on the information in the log. In this thesis, we focus on the control-flow perspective of the first type of process mining, i.e., process discovery.

In the process discovery category, 3 types of perspectives can be performed based on the information available in the event log. If the log contains information about activities executed to handle a particular case (i.e., process instance), the control-flow perspective could be discovered. This perspective is described to answer the question “How?”. If the event log provides information about the persons, systems, or machines that are involved in the execution of the activity, the organizational perspective can be derived; this allows to answer the question “Who?”, while the question “What?” is answered by mining the case perspective if the event log contains additional data associated with tasks.

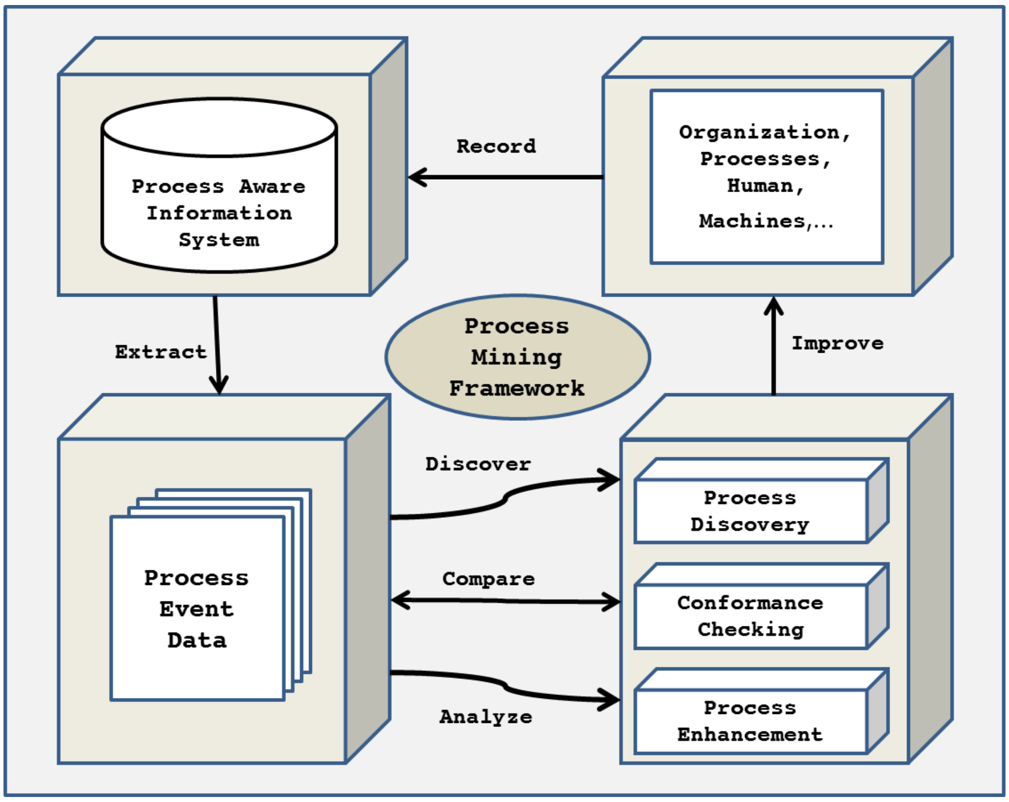


Figure 1.1. Overview of process mining and its three types of techniques

Table 1.1 illustrates an example of an event log of a purchasing process used for process mining. Each row in the table represents one event and each column represents an attribute of this event. Events are associated with cases, and in Table 1.1 the events are grouped by case and arranged chronologically. The first recorded event is related to case Q521-QZR and represents the execution of the activity *Purchase Request* performed by the employee *Hind* on *May 15, 2014*. Additional attributes related to this event can also be used by process mining such as data attributes entered, costs, etc. Each event is required to have a unique identifier. Each event should be related to a case and the events should be sorted. Basically, timestamps of execution are used to sort events chronologically. The timestamps shown in Table 1.1 represents the time when the execution of the corresponding activity was started. Timestamps which indicates the completion or the pause of the execution of an activity also can be recorded and used by process mining.

This log contains information about three cases (i.e., workflow instances). The log shows that for the three cases (1,2, and 3) the tasks *Request*, *Approve, Verify*, *Finalize*, *Order*, and *Receive & Verify* have been executed. Each case starts with the execution of Purchase

Request and ends with the execution of *Receive & Verify*.

Table 1.1. An example of an event log used for process mining

Case id	Event id	Activity	Timestamp	Resource	….
Q521-QZR	N0060	Purchase Request	5/15/2014 16:35	Hind	….
	N0070	Purchase Approval	5/15/2014 16:40	Safae	….
	N0080	Approval Verification	5/16/2014 16:40	Soukaina	….
	N0090	Purchase Finalization	5/16/2014 17:42	Younes	….
	N0100	Purchase Order	5/16/2014 17:30	Zineb	….
	N0110	Purchase Reception & verification	6/13/2014 10:55	Mohamed	….
Q523-B85	N0060	Purchase Request	5/15/2014 16:36	Hind	….
	N0070	Purchase Approval	5/15/2014 16:41	Safae	….
	N0080	Approval Verification	5/15/2014 16:41	Soukaina	….
	N0080	Approval Verification	5/15/2014 16:42	Soukaina	….
	N0090	Purchase Finalization	5/15/2014 16:55	Younes	….
	N0100	Purchase Order	5/15/2014 18:00	Zineb	….
	N0110	Purchase Reception & verification	6/15/2014 16:44	Mohamed	….
Q543-289	N0060	Purchase Request	5/15/2014 10:38	Hind	….
	N0070	Purchase Approval	5/15/2014 11:41	Safae	….
	N0080	Approval Verification	5/15/2014 12:42	Soukaina	….
	N0060	Purchase Request	5/15/2014 13:42	Hind	….
	N0080	Approval Verification	5/15/2014 15:42	Soukaina	….
	N0090	Purchase Finalization	5/15/2014 16:43	Younes	….
	N0100	Purchase Order	5/15/2014 16:45	Zineb	….
	N0110	Purchase Reception & verification	6/15/2014 11:45	Mohamed	….
. .	. .	. .	. .	. .	. . .

The event data shown in Table 1.1 shows typical information present in an event log. The first class of process mining which is process discovery can use the case id and event id of the event log shown in Table 1.1 to discover a process model which will represent the real behavior of the purchasing process in this example. The basic idea is to construct the model shown in Figure 1.2 from the information given in Table 1.1.

Conformance checking techniques will compare the process model discovered from the event data in Table 1.1 with a priori-model or the existing model. By comparing both models unconformities, deviations, etc. can easily be observed. Process enhancement can use for instance the timestamps associated with the events to investigate the performance of the process.

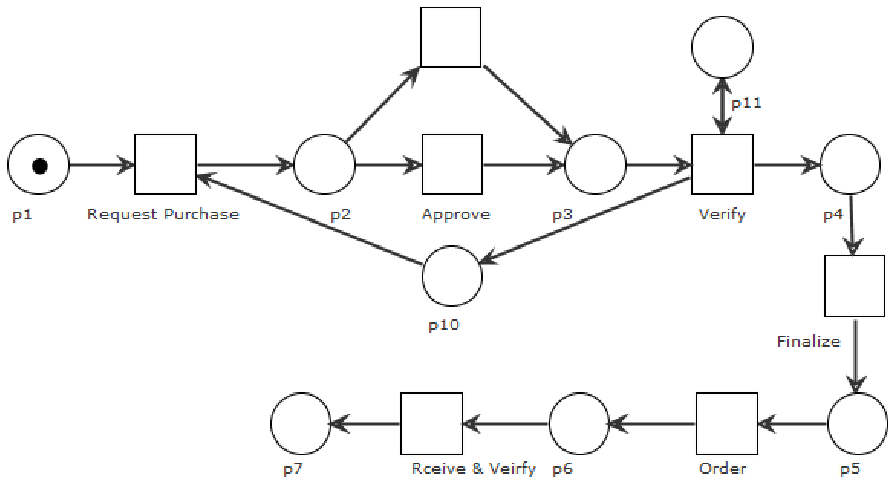


Figure 1.2. A process model corresponding to the event log of Table 1.1

1.3. Process Mining Challenges

Process mining became one of the hot topics in business process management and improvement. Therefore, the IEEE Task Force on Process Mining, under a group of more than 75 people and more than 50 organizations, created the Process Mining Manifesto to promote the topic of process mining. The Process Mining Manifesto define a set of guiding principles and list important challenges in order to offer a guide for software developers, scientists, consultants, business managers, and end users (IEEE Task Force, 2011). Although in this Manifesto a set of important challenges has been defined, none of the published research papers investigated whether the defined challenges of process mining have been solved or are still being encountered. Therefore, we conducted a comprehensive and critical review of the literature to identify the state of the art of process mining challenges. Through the literature review, this study aims to identify the most important issues of process mining that received considerable focus and those that still require attention, investigate whether these challenges are still encountering or have been solved, and summarize the technologies and approaches developed to deal with these issues and highlight their limitations if exist. We believe that the state of the art of process mining challenges will contribute to increasing the maturity of the field of process mining by underscoring the issues that have not been solved yet as well as the problems newly appeared while dealing with the main challenges of process mining. Still unsolved challenges and the problem that appeared in the attempt of solving the challenges can serve as a starting point for further researches. The Process Mining Manifesto (IEEE Task Force, 2011), published by the IEEE Task Force on Process Mining, lists several challenges and guiding principles for process mining. Table 1.2 outlines the main issues encountered in the area of business process mining (see Figure 1.3).

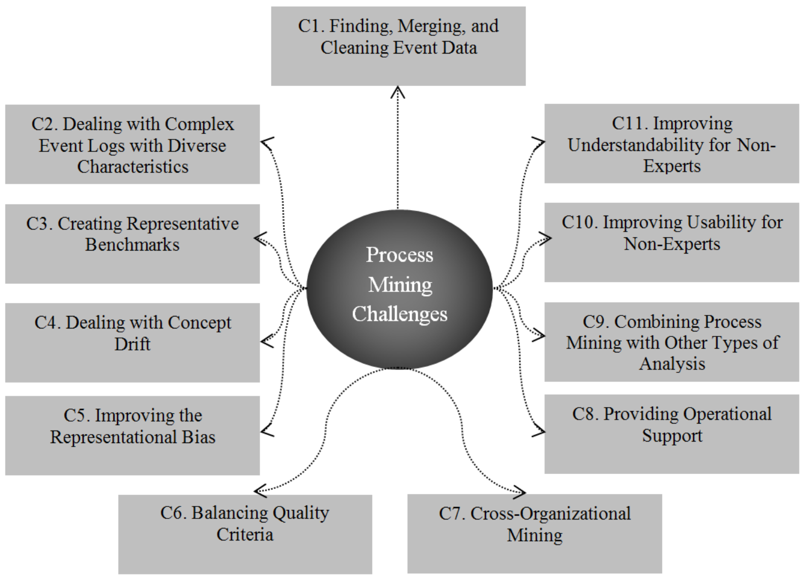

C1. Finding, Merging, and Cleaning Event Data
C2. Dealing with Complex Event Logs with Diverse Characteristics
C3. Creating Representative Benchmarks
C4. Dealing with Concept Drift
C5. Improving the Representational Bias
C6. Balancing Quality Criteria
C7. Cross-Organizational Mining
C8. Providing Operational Support
C9. Combining Process Mining with Other Types of Analysis
C10. Improving Usability for Non-Experts
C11. Improving Understandability for Non-Experts
Process Mining Challenges

Figure 1.3. Process Mining Challenges

Challenge C1: Finding, Merging, and Cleaning Event Data

Events (the starting point of process mining activity) can be stored in various and different data sources. Moreover, they might occur in a particular context, could be object centric rather than process centric, can be incomplete and noisy, and might also be characterised by different levels of granularity. The quality of a process mining result depends heavily on the input. Therefore, extracting event data suitable for process mining require considerable effort.

Challenge C2: Dealing with Complex Event Logs with Diverse Characteristics

Event logs can have very different characteristics (IEEE Task Force, 2011). Large event logs are difficult to handle, whereas small event logs do not provide enough data to make reliable conclusions.

Challenge C3: Creating Representative Benchmarks

A lot of process discovery techniques are available. However, a good benchmark to compare and evaluate different existing techniques and tools is missing.

Challenge C4: Dealing with Concept Drift

The concept drift means that the process might change over time while being analysed. Understanding concept drift is of prime importance for process management. Therefore, additional research and tool support are required to adequately analyse concept drift (IEEE Task Force, 2011).

Challenge C5: Improving the Representational Bias Used for Process Discovery

The representation bias used in the process discovery affect strongly the quality of process mining results. For this reason, it is necessary to be careful while selecting the appropriate representational bias.

Challenge C6: Balancing among Quality Criteria

It is challenging to judge the quality of a discovered process model while balancing the four quality criteria: fitness, simplicity, precision, and generalisation.

Challenge C7: Cross-Organizational Mining

Two types of cross-organizational process mining exist. The first type is when different organisations collaborate together to handle process instances. In the second one, different organisations follow the same process (IEEE Task Force, 2011). Existing process mining techniques deal only with a single organisation.

Challenge C8: Providing Operational Support

Process mining is not limited to analysis of historical data but could also be applied to online operational support (detect, predict, and recommend). The challenge is to handle computing power and data quality issues while applying process mining techniques in such online setting.

Challenge C9: Combining Process Mining with Other Types of Analysis

The challenge is to combine process mining approaches with other types of analysis techniques to better extract more information from event data.

Challenge C10: Improving Usability for Non-Experts

End users need to interact with the results of process mining. For this reason, hiding the sophisticated process mining algorithms behind user-friendly interfaces is necessary.

Challenge C11: Improving Understandability for Non-Experts

The purpose of process mining is to produce process models that can be used for further analysis and not for documentation (Buijs, 2014a). Therefore, the understandability of the results for non-experts should be improved.

1.4. The state of the art of process mining challenges

In order to identify publications that address process mining challenges we investigated three platforms: SCOPUS database, Process Mining Wiki, and google scholar. SCOPUS is the largest database of peer-reviewed literature. Process Mining Wiki is a publication platform that promotes research on the topic of process mining and contains publications only on process mining field. It was created by Process Mining Group of Mathematics and Computer Science Department of the Eindhoven University of Technology in the Netherlands, chaired by Will van der Aalst. Moreover, to not miss any paper, we also explored Google Scholar which allows a wide range of academic literature. Since Process Mining Manifesto released at the end of 2011, we focused on the papers published from the beginning of 2012 till the beginning of 2017. Using the three databases, 105 publications dealing with process mining challenges have been identified. We conducted a comprehensive and critical review of these papers to identify the state of the art of the main issues of process mining, specifically to investigate whether the reported challenges are still encountering or have been solved, identify the challenges that received considerable focus and those that still require attention, and summarize the technologies and approaches developed to deal with these challenges and outline their limitations if exist. Figure 1.4 depicts the number of papers dealing with each problem of process mining that we identified from the three publication databases. The most commonly addressed problems were Finding, Merging, and Cleaning Event Data (19 papers) and Dealing with Complex Events (15 papers). These were followed by Combining Process Mining with Other Types of Analysis and Improving Usability for Non-Experts (13 papers each). A substantial number of papers also addressed the problem of Dealing with Concept Drift (11 papers). The challenge of Cross-Organizational Mining came at the 5th position with nine publications. A small number of papers focused on Balance among Quality Criteria and Providing Operational Support (6 papers each). Likewise, the problems of Creating Representative Benchmarks (5 papers), Improving the Representational Bias for Process Discovery (4 papers) and Improving Understandability for Non-Experts (4 papers) have not been thoroughly researched.

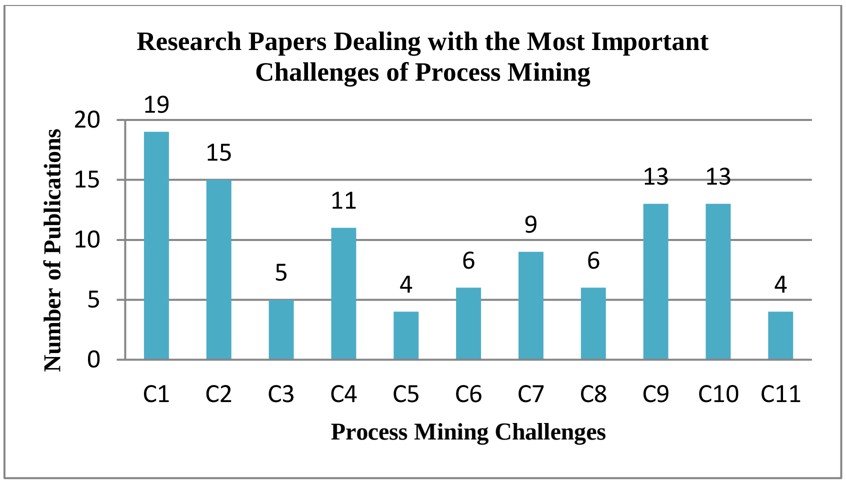


Figure 1.4. Publications dealing with the main process mining challenges

Based on the conducted review, we found that a number of challenges encountered by process mining researchers have been partly or fully addressed in the past five years. One of the most frequently addressed problems encountered in process mining is Finding, Merging, and Cleaning Event Data, closely followed by Dealing with Complex Events. Other well-researched areas include Combining Process Mining with Other Types of Analysis and Improving Usability for Non-Experts, closely followed by Dealing with Concept Drift and Cross-Organizational Mining. However, the review highlighted a number of process mining challenges that still need to be addressed further. A small number of papers focused on Balance among Quality Criteria and Providing Operational Support. Likewise, the problems of Creating Representative Benchmarks, Improving the Representational Bias for Process Discovery and Improving Understandability for Non-Experts have not been thoroughly researched.

1.4.1 Finding, Merging, and Cleaning Event Data

Nooijen *et al.* (2012) presented an automatic technique of discovering the model of a data-centric process. For each data object in the process, a separate process model that describes the evolution of this object (also known as an artifact life-cycle model) is discovered. However, the user needs to choose the number of artifacts (k) to be discovered. Therefore, it is required to develop more automatic approaches to identify the right number of artifacts (k).

Ly *et al.* (2012) proposed the concept of semantic log purging, i.e., cleaning logs based on domain-specific constraints using the knowledge that typically complements processes for the purpose to enhance the quality of a mined process model. The approach actually cleans the log with respect to its expected properties which is our major focus in this challenge.

Leemans, S et al. (2014b) presented a technique called inductive miner infrequent able to discover process models from event logs containing infrequent behavior. The technique filters infrequent behaviour using the eventually-follows graph.

Mannhardt *et al*. (2014) presented a method to extend an incomplete main event log with events from supplementary data sources, even supposing references to the cases recorded in the main event log are missing. The method can be used to correlate a large portion of events and also could be used to connect supplementary events to traces of the main event log without violating the constraints on different perspectives.

Leemans, S. *et al.* (2014a) introduced a robust process discovery algorithm based on the inductive miner and on probabilistic behavioural relations which are less sensitive to incompleteness. The algorithm is proven by experiment to be capable of rediscovering models from small incomplete event logs and requiring less information in the event log compared to other process discovery algorithms.

Priyadharshini and Malathi (2014) proposed new methods for noise filtering problem. They used unsupervised noise filtering and frequent group-based noise filtering algorithms for this purpose. The performance of the proposed methods was better than other existing methods.

Claes et al. (2014) proposed a rule-based technique to merge the data at a structured level of various partners involved in a cross-organisational process. The technique consists of an approach for configuring and executing the merge and an algorithm for discovering links between the data of the different partners.

Van der Aalst (2014a) described the idea of extracting “flat event logs” from existing databases. Flat event logs are necessary for using traditional process mining techniques. Therefore, the author described an approach that scopes binds, and classifies data for this purpose. However, the author only conceptualised the idea; a development of support tools is required.

Cheng and Kumar (2015) performed extensive experiments to evaluate how a rule-based approach called log sanitisation could improve the performance of a process mining algorithm. The proposed technique for sanitising noisy logs consists first of building a classifier on a subset of the log data and then applying the rules of the classifier in order to remove noisy traces. This general approach for log sanitisation performed good results and can work with any noise pattern.

Leoni *et al.* (2015b) described the notion of log–model alignment and certain diagnostics that can be computed using alignments tailored towards declarative models. The alignment-based approach can be used for cleaning event logs to remove traces unnecessary for further analysis.

Wang *et al*. (2015) presented a graph repair approach for cleaning event logs. The method repairs; based on a reference process model, an event log that contains events with inconsistent labels (labels which do not correspond to the labels of the reference model). Nevertheless, this approach necessitates the availability of a reference process model which is usually unavailable.

De Murillas *et al*. (2015) proposed a generic method that uses so-called redo logs that several Data Base Management Systems (DBMSs) adopt for data integrity and recovery reasons. Since such logs contain a list of events but are not correlated in the form of traces, the proposed approach convert these logs into event logs by using the relations expressed in the data model of the DBMS.

Calvanese et al. (2015) presented a methodology that aids domain experts to flexibly extract XES event logs from legacy relational databases. The extraction is based on a conceptual representation of the domain of interest in terms of ontology. This ontology is associated with the underlying legacy data leveraging the ontology-based data access (OBDA) paradigm. The framework allows one to enrich the ontology via user-oriented log extraction annotations, which can be flexibly exploited to provide different log-oriented views over the data.

Leemans, M *et al*. (2015) proposed a reverse engineering based methodology for extracting event logs from distributed software systems, crossing multiple system components. The approach is language independent and is restricted to single-threaded distributed software systems.

As events which do not directly refer to the executions of activities recognizable for process operators are not appropriate for process mining, and the events recorded in an information system usually do not directly refer to recognizable activities, Mannhardt et al. (2016) proposed a supervised event abstraction method that provides a mapping from the recorded events to activities recognizable by process operators. The abstraction method is based on behavioural activity patterns that extract domain knowledge on the relation between high-level activities and recorded low-level events. The approach results in an abstracted event log providing insights that cannot be obtained with an original event log when applying process discovery techniques.

Lu et al. (2016a) introduced an integrated tool named Log to Model Explorer that allows users; in an interactive and iterative way, exploring and pre-processing a log by clustering, filtering infrequent events and renaming event label.

Events granularity of event logs might be too low level. Thus, event logs with an appropriate level of event abstraction are required. Tax et al. (2016) described a supervised learning based method for abstracting events in an XES event log that is too low-level. The technique consists of producing a feature representation of events based on XES extensions, and of a Conditional Random Field based learning step to abstract events in an event log. The method enables process discovery algorithms to construct processes with high-level insights.

Conforti et al. (2017) proposed an automated technique for removing infrequent behaviour (noise) from event logs. The approach consists first of constructing an abstraction of the event log as a directed graph that captures direct follows dependencies between event names. Then, infrequent transitions are erased from this directed graph. After that, the original log is replayed on the updated directed graph and finally, unfitting events are erased from the event log.

Suriadi et al. (2017) described, in the context of cleaning event logs, a pattern-based approach that provides a repository of knowledge to handle data imperfections encountered when preparing event logs. The pattern-based approach offers a documentation that collects typical issues which can be faced in preparing event logs for the purpose of process mining use as well as associated remedial actions to repair them.

Because the quality of any process mining technique result strongly depends on event logs, finding, merging and cleaning event logs was reported as the first challenge in Process Mining Manifesto and requires considerable efforts. In order to handle this challenge, researches have to deal with event log stored in various data sources, event data that had been executed in a certain context, event data that is object centric than process centric, incompleteness, noise (infrequent behaviour), and event data characterised with different levels of granularity. As can be seen, all these elements have been directly and indirectly examined in the publications cited above. And we can say that the first challenge of Process Mining Manifesto has received a significant focus from the research community. Table 1.2 summarises the methods used in dealing with this challenge.

Table 1.2. Summary of the approaches used to deal with finding, merging, and cleaning event data

Paper Ref.	**Used methodology**	**Outcome**
Nooijen *et al.* (2012)	Automatic technique	Discover a data centric process model (artefact life-cycle model)
Ly *et al.* (2012)	Domain specific constraints based approach	Clean the log with respect to expected properties
Leemans, S. *et al.* (2014a)	Inductive Miner and probabilistic behavioural relations based algorithm	Rediscover process models from small incomplete event logs
Leemans, S. *et al.* (2014b)	Eventually-follows graph based technique	Filter infrequent behaviour and return sound model
Claes et al. (2014)	Rule based technique	Merge historical data of different partners in one event log
Mannhardt *et al.* (2014)	Alignment-based method	Ensure correlation between supplementary and main event logs
Priyadharshini *et al.* (2014)	Unsupervised and frequent group-based noise filtering approach	Filter noise from event logs
Van der Aalst (2014a)	Scope, bind and classify based approach	Extract "flat event logs" from existing databases
Cheng *et al.* (2015)	Noisy log sanitisation based approach	Filter noise and improve the performance of a process mining algorithm
Leoni et *al.* (2015b)	Alignment based approach	The approach removes traces that should not be used for further analysis
de Murillas *et al.* (2015)	Generic approach	Convert redo logs of DBMSs into event logs
Leemans, M *et al.* (2015)	Reverse engineering based technique	Extract event logs from distributed systems
Calvanese *et al.* (2015)	Ontology based framework	Extract event logs from relational databases
Wang *et al.*	Structure-based graph repair	Clean event logs by repairing

(2015)	approach	inconsistent event names
Mannhardt *et al*. (2016)	Supervised event abstraction method	Generate an abstracted event log better than the original one
Tax *et al*. (2016)	Supervised learning based method	Abstract events in an event log
Lu *et al*. (2016a)	Log to Model Explorer tool	Filter infrequent events
Conforti *et al*. (2017)	Automated technique	Remove infrequent behaviour from event logs
Suriadi *et al*. (2017)	Pattern-based approach	Extract data imperfection and provide associated remedial solutions

1.4.2 Dealing with complex event logs with diverse characteristics

Van der Aalst (2013a) introduced a general decomposition technique for process discovery and conformance verification through which large process mining problems are decomposed into a set of smaller problems. However, in conformance checking decomposition, only fitness was considered, whereas other quality dimensions; such precision and generalisation; are also important and need to be investigated. Regarding process discovery decomposition which was based on splitting all activities into overlapping activity sets, large scale experimentation is required.

Van der Aalst (2013b) proposed a general divide-and-conquer technique in which an event log is decomposed based on splitting activities without focusing on a specific representation. The approach allows for various decomposition strategies. Moreover, it provides new and interesting insights about the essential requirements for decomposing process discovery and conformance checking issues. This approach has been later extended and implemented by (Verbeek et al., 2016a) in a tool framework called Divide and Conquer. This tool allows (1) easy decomposed discovery using six discovery algorithms, and (2) the end user to select the classifier to use, the discovery algorithm to use, and a configuration to use.

Van der Aalst (2013c) formalised the notion of process cubes that organises events and process models using different dimensions. Each cell in the process cube corresponds to a set of events which can be used to apply process mining techniques without any extraction of event logs beforehand. However, many challenges remain and need to be addressed in further researches, such as comparing and visualizing different cells, computing sub-logs and models per cell, and addressing concept drift.

Munoz-Gama *et al*. (2013) proposed a conformance checking technique that decomposes large processes into small processes using the so-called Refined Process Structure Tree (Polyvyanyy et al, 2010) which allow the construction of a hierarchy of Single-Entry Single-Exit. However, more real life case studies are required to prove the correctness of the approach.

The work of Munoz-Gama *et al.* (2014) presented Single-Entry Single-Exit (SESE) decomposition technique for conformance checking where large process models and event logs are divided into smaller fragments that can be analysed independently. The event log is simply partitioned into sub-logs in a way that every trace appears exactly in one sub-log. The approach provides rapidly detailed and focused diagnosis of conformance checking. However, none of the quality dimensions was considered.

Van der Aalst and Verbeek (2014b) proposed passages based approach where process mining problems are decomposed into smaller problems. Both process discovery and conformance checking can be partitioned using passages. The discovered fragments can be merged into an overall process model while the result of conformance checking per passage can be incorporated into overall conformance diagnostics. Using such decomposition the two types of process mining perform efficiently. However, the performance of the results strongly depends on the number of passages used and the size of the large passages. Large scale experiments are needed in further works.

Hompes *et al.* (2014) proposed a process discovery decomposition approach based on clustering large event logs into sub events (activities). Moreover, three quality notions based on clustering properties have been defined: cohesion, coupling, and balance to assess a decomposition before it is used to discover a model or check conformance. The decomposition technique was based on these quality dimensions. However, it is challenging to choose the suitable discovery algorithm that can be applied to the decomposed event log.

Kalenkova *et al*. (2014b) presented a process discovery decomposition approach that partitions transition system into parts based on the knowledge about the event log, and to each part, one of the regions based discovery algorithms are applied. The small discovered models are at the end merged into one process model well-structured and more readable. However, generalisation quality dimension has not been considered in this study.

Verbeek and Van der Aalst (2015) introduced a generic framework for decomposed discovery and replay that divides big event log containing many activities into a set of small event logs. Then the authors applied Integer Linear Programming (ILP)-based algorithms to the decomposed process discovery and conformance checking. The resulting ILP-based discovery algorithm was much faster than the regular discovery algorithm, but the resulting ILP-based replay algorithm was more complex. However, the framework supports only the α-algorithm and the ILP-based algorithms. Much more works are needed to extend this approach to include other algorithms.

Leemans, S. *et al*. (2015a) introduced Inductive Miner - directly-follows based (IMD) framework that applies divide and-conquer strategy to discover a process model from event log containing billion of events. The approach consists in selecting the cut of process activities, using this cut to divide the event log into sub blogs, and recursing the sub logs on until a base case is confronted. Also, a new Projected Conformance Checking framework was developed to evaluate the quality of the mined model that is discovered from a very big event log but only fitness and precision metrics are considered.

Users need to operate easily the decomposition techniques for process discovery and conformance checking, from splitting a large problem into small problems to merging the results into a single result. For this purpose, Verbeek (2014, 2016a) introduced the *DivideAndConquer tool*, and Vogelgesang and Appelrath (2015) presented the *PMCube Explorer*.

Verbeek et al. (2016b) proposed an algorithm to merge decomposed alignments, resulting from replaying decomposed logs on associated decomposed nets, into an overall alignment. In case it is not possible, the algorithm returns a pseudo-alignment, i.e., a relaxation of the normal alignment. The algorithm consists of three alignment rules and two pseudo-alignment rules.

Verbeek et al., (2017) showed that there are some cases where the decomposed replay requires much more time compared to the monolithic replay (without decomposition). To alleviate this problem, they proposed hide and reduce decomposed replay (hide transitions and reduce nets). Although in some other cases, this alternative abstraction also takes more time than the original decomposed reply and monolithic reply, it handles more situations than other replays do, and the replay cost estimate is better than the original decomposed replay.

Traditional process mining techniques have difficulties to handle “big event data” properly. Fortunately, it is generally agreed that decomposed process mining is the best solution to this question, as it decomposes the process mining problem into many smaller problems that can be solved in short time, and many ways to partition process mining problems exist. However, the approaches used for decomposition were not consistent among studies: some used the divide-and-conquer approach, some used the Single-Entry Single-Exit technique, some used the notion of process cubes, and others have based their decomposition on clustering. Hence, an evaluation of these approaches is needed for selecting the appropriate technique and identifying the conditions under which each approach should be used. Moreover, each study for process mining decomposition has some limitations and need to be investigated in further works. The most relevant problem was balancing the four conflicting quality dimensions, especially in conformance checking decomposition. Not all quality metrics have been considered in their studies. Furthermore, some studies still have issues related to computation and visualisation. Table 1.3 provides a summary of the frameworks that have been developed to deal with complex and large event logs.

Table 1.3. Summary of the approaches used to deal with complex event logs with diverse characteristics

Paper Ref.	**Used methodology**	**Outcome**	**Limitation**
Van der Aalst (2013a)	Generic decomposition technique based on a set of activities	Allow large process mining problems to be decomposed into set of smaller problems	Only fitness metric has been considered in conformance checking
Van der Aalst (2013b)	General divide-and-conquer approach based on a partitioning of activities	Allow for several decomposition strategies	
Van der Aalst (2013c)	Process cubes based approach	Enable users to analyse and explore processes interactively on the basis of a multidimensional view on event data	Remaining issues: - Visualising cells. -Computing sub logs and models per cell. -Concept drift
Munoz-Gama *et al.* (2013)	Conformance checking decomposition technique based on Refined Process Structure Tree	Provide efficiency in computation and diagnosis when discovering conformance problems	
Munoz-	Single-Entry Single-Exit	Provide rapidly detailed	Quality dimensions

Gama *et al.* (2014)	decomposition technique for conformance checking	and focused diagnosis of conformance checking	was not considered
Van der Aalst *et al.* (2014b)	Passages based decomposition approach	Conformance checking and process discovery can be done much more efficiently	The performance depends on the number of used passages
Hompes et al. (2014)	Discovery decomposition technique based on clustering	Provide efficient decomposition	Choosing the appropriate algorithm to mine the decomposed event logs is a challenge
Kalenkova *et al.* (2014b)	Process discovery decomposition from transition system	Discover better structured and more readable process models from transition systems	Generalisation metric has not been considered
Verbeek (2014a)	DivideAndConquer tool	Enable users to operate easily decomposition techniques	
Leemans *et al*. (2015a)	Inductive Miner - directly-follows framework & Projected Conformance Checking framework	Enable users to discover sound model from an event log with billions of events	Generalization metric has not been considered
Verbeek et al. (2015)	Generic decomposition for discovery and reply + Integer Linear Programming based algorithm	The ILP-based decomposed algorithm is much faster than the regular discovery algorithm. ILP-based replay algorithm is more complex.	Support only the α-algorithm and the ILP-based algorithm
Vogelgesang *et al.* (2015)	PMCube Explorer	Enable users to operate easily decomposition techniques	
Verbeek *et al. (*2016a)	Divide and Conquer tool framework	Enable users to operate easily the decomposed discovery and replay	

Verbeek *et al.* (2016b)	Decomposed alignments merging algorithm	Merge decomposed alignments into an overall alignment or a pseudo-alignment	Pseudo-alignment may not better match than the overall alignment
Verbeek *et al.* (2017)	Hide-and reduce decomposition approach for replay	Provide a better estimate of the replay cost and handle more situations than other replays	The speed is slower than the original decomposed replay in some situations

1.4.3 Creating representative benchmarks

Rozinat, A., et al. (2007) have written an important and a good starting article on building process mining algorithms benchmark framework. They introduced two methodologies to evaluate process discovery techniques. The first strategy is based on evaluation using metrics and the second one is a machine learning strategy. Wang, J., et al. (2012) extended this work by empirically evaluating process discovery techniques using both artificial and real-life datasets, as well as different similarity measures. Moreover, Gupta (2014) provided a comparison of Heuristic miner, Fuzzy miner, and Genetic miner algorithms to identify in which case these algorithms can be used. Despite the fact that these frameworks allow users and businesses select the process mining algorithm suitable to a given event log, empirical evaluation is time-consuming. Only after performing experiments on all existing mining algorithms can be decided the best algorithm. Wang, J. et al. (2013) presents particularly remarkable work in which the problem of spending a long time in performing experiments to select the appropriate mining algorithm at the end is solved. Their proposed framework is based on learning step and recommendation step. In the first phase, a regression model is built based on features extracted from high quality selected reference models and on the similarities between the reference models and the mined models. Using this regression model as well as features extracted from the other models, similarities between reference models and mined models are predicted without performing experiments. While the authors' results obtained by applying this framework are very accurate and attractive, there is a weakness in this concept. The presented methodology is strongly based on reference models. However, reference models usually are not available in the practical world. Ribeiro, J. et al. (2014) developed a framework and a tool for recommending control flow algorithms. Based on features extracted from event logs and prediction models built from experiments, top-K control-flow miners are recommended. Although this work takes event logs as input rather than reference models considering the limitation in (Wang, J. et al., 2013), the study can be criticised on the fact that in this framework, performing experiments to build prediction models are required before recommending top-k mining algorithm. Thus, again time-consuming. The proposed systems would have been more interesting if the authors had based their frameworks only on event logs as an input and at the same time had reduced the experiments required to decide the mining algorithm suitable for a given event log. Pérez-Alfonso et al, (2015) proposed an approach to recommend a process discovery algorithm based only the classification of event logs, but they just conceptualised the idea.

1.4.4 Addressing Concept Drift

Bose *et al.* (2014) introduced the topic of concept drift in process mining and proposed a generic framework and a set of features for adequately detecting changes in event logs and localising changes in a process. They have demonstrated that by using the concept drift system, heterogeneity among cases caused by process changes can be effectively detected.

The work of Rekhadevi and Appini (2015) described an idea float based framework and specific strategies for identifying when procedure changes; and limiting the parts of the procedure that have changed. They demonstrated that process changes can be managed if the idea floats have been determined.

Nithya *et al.* (2015) used the drift concept to determine agent guilt. They proposed a framework based on data strategies across the agent to upgrade the likelihood of determining if there is a leak of data. Dealing with concept drift system presented in this paper can be used to identify changes in real life event logs even with an insignificant number of cases.

Raviteja Pochiraju and Kumar (2015) proposed a generic approach and particular strategies for identifying the parts of the process that have been changed once a method is modified. The framework is based on various area units that characterise relationship among activities to discover variations.

Aruna and Laxmi Priya (2015) introduced the first online procedure to identify and handle the drift concept. The proposed system stands on using both abstract interpretation and sequential sampling with the new data stream approaches. Their results also demonstrated that it is possible to efficiently handle non homogeneous cases generated by process changes.

Li and Kang (2015) proposed new process mining procedure to rebuild the workflow process that faced deviations of workflow instances. The system consists of building MarKow transition matrix based on analysing the workflow log, and then in developing a multi-step workflow mining algorithm to discover structurally relationships between activities. The approach has been proved to be applicable.

Hompes et al. (2015a) proposed a trace clustering approach based on the Markov cluster (MCL) algorithm for detecting common and deviating behaviour based on a set of selected perspectives. In this technique trace clustering and outlier detection are combined in order to find mainstream and deviating behaviour. The process context is considered by using both control-flow and case data in order to be able to find and explain both common and exceptional behaviour. However, MCL algorithm is non-parametric in the number of clusters. So, the expansion and inflation parameter is set manually. This work was extended in (Hompes *et al.*, 2015b) by providing a comparative trace clustering method that is capable of detecting changing behaviour in a process by using both control-flow and case data. The approach consists of comparing clusterings constructed for two selected fragments of an event log to detect change point. The comparison includes differences in behaviour over time as well as for distinct case groups, i.e., cases handled by different resources.

Lu et al, (2016b) proposed mappings between events based method to detect deviating events by identifying frequent similar behaviour and dissimilar behaviour among executed process instances, without discovering any normative model.

Kakkad & Sheikh, (2016) proposed a generic framework to analyse process changes based on events logs. The framework consists of different features sets that characterize relationship among activities in the event log to detect the changes and identify the regions of change in a process.

Sethi & Kantardzic (2017) presented the Margin Density Drift Detection (MD3) algorithm, which is able to accurately detect concept drift from unlabeled streaming data. This algorithm exploits the number of samples in a classifier’s region of uncertainty (margin), as a metric for detecting drift. It is robust to stray changes in data distribution, a reliable substitute to supervised drift detectors, and also can be used in a variety of data stream environments.

The papers cited above proposed solutions for dealing with concept drift.

Nevertheless, most of the works considered changes only from the control-flow perspective except Hompes et al. (2015a, 2015b), whereas the data and resource perspectives are equitably essential to gain more insights. Hence, more methods which allow detection of changes from other perspectives need to be established. Moreover, drift detection was performed only in an offline setting, but it is also very important for online analysis. In addition, while working on drift concept, researchers faced some issues that need to be addressed. See Table 1.4.

Table 1.4. Summary of the approaches used to deal with concept drift

Paper Ref.	**Used methodology**	**Outcome**	**Limitation**
Bose *et al.* (2014)	Generic framework and set of features	Detect changes in event logs and localise changes in a process	- Control-flow perspective only - Encountering challenges: 1. Change-pattern specific features. 2. Feature selection. 3. Holistic approaches. 4. Recurring drifts. 5. Change process discovery. 6. Sample complexity. 7. Online(on-the-fly) drift detection.
Rekhadevi *et al*. (2015)	Idea float based framework	Process changes can be managed with the identification of idea floats	
Raviteja *et al.* (2015)	Generic approach and particular strategies	Discover variations	
Aruna *et al.* (2015)	Online procedure	Handle efficiently non homogeneous cases generated by process changes	
Li *et al.* (2015)	Process reconstruct approach based on the Markov transition matrix of event log	Rebuild the workflow process that faced deviations of workflow instances	
Nithya *et al.* (2015)	Agent guilt identification based framework	Determine changes in real life event logs even with insignificant number of cases	Control-flow perspective only
Hompes *et al.* (2015a)	Markov cluster algorithm based trace clustering approach	Detect mainstream and deviating behaviour	The expansion/inflation parameter of the MCL algorithm is set manually
Hompes et	Comparative trace	Detect differences in	Analysis process

al., (2015b)	clustering approach	behaviour	automation and changes visualization are required
Kakkad *et al.* (2016)	Generic framework and set of features	Detect and localize the changes in a process	Control-flow perspective only
Lu *et al.* (2016b)	Mappings between events based approach	Detect deviating events without discovering a normative model	The approach accuracy is slightly lower when deviations are frequent and more structured. Control-flow only
Sethi *et al.* (2017)	Margin Density Drift Detection (MD3) algorithm	Accurately detect concept drift from unlabeled streaming data	Detect drifts with significantly fewer false alarms. Control-flow only.

1.4.5 Improving the Representational Bias Used for Process Discovery

Buijs *et al*. (2012a) presented a genetic process discovery through which process trees are mined. The proposed technique assures the model's soundness and is the first to guarantee the correctness of the model while including all four quality criteria.

Mokhov and Carmona (2014) described the first try to use Parameterized Graphs (PGs) in process mining field. Although they showed the potential benefits of using Parameterised Graphs, there are still encountering issues such as handling cyclic behaviour, reaching scalability, and producing BPMN diagram that requires being addressed.

Kalenkova *et al.* (2015) developed various and powerful control flow transformation techniques to mine BPMN diagram (preferred by businesses) from famous control flow modelling representations such Petri nets, causal nets, and process trees while taking into account the representational bias.

Swapnali and Ravi (2015) focused on resolving the current issues related to process discovery techniques, such as the incapacity to mine precise and understandable process models. Hence, they proposed ActiTraC algorithm based framework through which mined models can be produced automatically from systematic event logs.

Most research papers have focused on understandability, correctness, and quality of the representation and also on transformations techniques that convert control flow modelling representations to the desired language visualisation. However, the focus in this challenge should be toward the implicit search space implied by the representational bias (Van der Aalst, 2011b). For instance, BPMN notation does not support non-free-choice-constructs. Thus, for an event log containing non-free choice constructs, choosing BPMN notation is not the suitable representation of the process model discovered from this event log. From this latter, we can conclude that the characteristics of event log strongly impact representational bias. Therefore, in our opinion, if we know beforehand the characteristics of a given event log, selecting the right representation bias would be very clear from the beginning. Thus, frameworks that identify event log characteristics (i.e. non-free-choice construct, duplicate tasks, invisible tasks…) without using mining algorithms are required.

1.4.6 Balancing between quality criteria of fitness, simplicity, precision, and generalization

Buijs *et al* (2012a) proposed a new genetic process mining algorithm that discovers process trees from event logs. They demonstrated that it is possible to balance the four quality criteria. The presented technique is the first to guarantee correctness while including the different quality dimensions.

Fahland and Van der Aalst (2012) presented a post-processing method through which mined process models are simplified while adjusting overfitting and underfitting. The technique presented in this paper can be associated with any process discovery algorithm that generates a Petri net able to reproduce the event log.

Buijs *et al.* (2013a) described the Evolutionary Tree Miner (ETM), a genetic process discovery algorithm through which users can control, according to their preferences, the process discovery with respect to the four quality criteria. It has been proven that all criteria are essential for process discovery. However, precision, generalisation, and simplicity depend heavily on the state of the fitness.

Müller *et al.* (2013) presented a genetic mining technique that discovers service model from a given service and the event log containing the execution of the given service. Moreover, the proposed algorithm balanced the four quality dimensions. Nonetheless, the obtained service model has the highest quality but not the optimal quality. Thus, an extension of this framework in further research to select the service model with optimal quality is required.

Van Eck *et al.* (2014) extended the work of Buijs *et al.* (2013a) by exploring a technique that enhances the performance of the ETM algorithm. The approach consists first of creating an initial population of the process model with acceptable quality and then in detecting and handling the quality problems in the model using guided mutation operations. This method allows generating models with high quality in fewer generations than the original ETM. However, the used mutation operations are not excellent. Using random mutation operations in further research is needed.

Van Zelst *et al.* (2015) introduced a sequence-encoding filtering framework where irregular behaviours are filtered while applying ILP-based process discovery algorithm. The proposed method allows generating less overfitted, and more comprehensible process models, and also is able to catch the frequent behaviour in the event log.

Whereas some works were not able to balance the four conflicting quality dimensions, others demonstrated that balancing the quality dimensions is possible. For instance, it has been proven that process discovery algorithms that generate process trees are able to balance the four quality dimensions. However, not all existing process discovery techniques can actually produce process trees. Therefore, it is recommended to develop techniques that allow the balance of the four quality criteria for the algorithms that discover other representations such as Petri net, causal net, and BPMN; or design a method like the one Fahland and Van der Aalst (2012) proposed, but it should not be restricted to algorithms discovering one specific notation. Table 1.5 provides a summary of the methods conducted for balancing between quality criteria.

Table 1.5. Approaches used to balance the quality dimensions of discovered process model.

Paper Ref.	**Used methodology**	**Outcome**	**Limitation**
Buijs *et al.* (2012a)	Genetic process mining algorithm to discover process trees	Ensure correctness while incorporating all four quality dimensions.	Process tree only
Fahland *et al.* (2012)	Post-processing approach	Simplify discovered process models while controlling the balance between overfitting and underfitting	Support only Petri net
Buijs *et al.* (2013a)	Genetic process discovery algorithm: Evolutionary Tree Miner	Allow users to control the process discovery with respect to the four quality criteria	Precision, generalisation, and simplicity depend on fitness
Müller *et*	Genetic service	Produce service models with	The high quality

al. (2013)	discovery algorithm	high balanced quality	of the mined service model might not be optimal
Van Eck *et al.* (2014)	ETM algorithm enhancing technique	Produce process models with high quality in fewer generations than the original ETM	The used mutation operations are not perfect
Van Zelst *et al.* (2015)	Sequence encoding filtering technique	Produce process models less over-fitting, more understandable, and more adequate in capturing the dominant behaviour in the event log	Only precision, simplicity, and fitness are balanced

1.4.7 Cross-Organizational mining

Buijs *et al.* (2012b) explored an original framework in which collections of process models can be compared with their events logs across organisations. The method is based on three types of metrics: metrics related to process models, metrics related to process executions, and metrics to compare process models and/or process executions. The authors demonstrated that even simple metrics offer useful insights regarding how to enhance processes. However, this method approach is generic and any metric can be used.

Buijs *et al.* (2013b) presented and compared four approaches to discover a configurable process model from a collection of event logs by extending the ETM genetic algorithm. They demonstrated that mining one configurable process model with commonalities and differences among variants is better than discovering one process model per organisation. This was the first paper in which a configurable process model was constructed based on a collection of event logs.

Zeng et al. (2013) introduced a process mining approach to discover the coordination patterns between various organisations and the process model of each organisation for cross-organizational workflow from the distributed running log. Since this log contains information about resource allocation, an RM-WE-Net model is proposed to represent the process model mined. Based on the model discovered for each organisation and the coordination patterns, a process integration approach is then proposed to obtain the model for a cross-organizational workflow. Nonetheless, the approach cannot handle some special structures such as choices or invisible tasks for one single organization.

Buijs and Reijers (2014b) proposed a comparison procedure of similar business processes across multiple organisations based on the alignment between registered behaviour and modelled behaviour. Their analytical procedure provides the possibility to analyse the actual execution of a process within a particular organisation with its intended model, and also with the variants of the same model used by other organisations.

Schunselaar *et al.* (2015a) offered an implementation of YAWL into the cloud. Their implementation allows multiple non-competitive organisations to help each other, and provide through configurable process models the opportunity to support different variants of the same process.

Sebu and Ciocarlie (2015) provided, after deep analysis of multiple inter-organisational cooperation modes, a process-based approach for determining compatibility between organisations. The method stands on using process mining algorithms and graph comparison methods to determine the most suitable organisations for profitable collaborations.

Yilmaz & Karagoz (2015) developed an environment where cross-organizational process mining is applied with the unsupervised learning in which predictor variables related to performances of organisations are used. The proposed approach consists of (1) mining the process models of organisations, (2) computing performance indicators, (3) clustering organisations based on performance indicators, and finally (4) underlining discrepancies between process models to make recommendations. But, Analyzers can fail in case there are loops in the process models.

Burattin et al. (2015) proposed a possible approach toward a complete solution to support the Cross-Organizational process mining while preventing the confidentiality of the dataset and processes. The framework uses AES as a symmetric cryptosystem for strings, and Paillier for the homomorphic encryption of numerical values. Unfortunately, the available analysis plugins in ProM do not involve numerical data attributes except conformance checking plugin to support the proposed framework.

Aksu et al. (2016) proposed a generic Cross-Organizational Process Mining Framework to accurately comparing organisations based on the usage of a software product such as Enterprise Resource Planning (ERP). The framework considers as input an event log, semantics, i.e., the meaning of terms in an organisation, and organisational context, i.e., the characteristics of an organisation. Through these inputs, the methodology is able to identify what to compare between organisations and how.

We can clearly observe that several techniques have been applied in many papers to deal with the challenge of cross-organizational mining. Most of them focused on commonality and collaboration between organisations, specifically on similarities between the process models and behaviou of organisations. However, although these approaches provide the way to successful cooperation, organisations might refuse such collaborations to avoid leakage of private information. Burattin et al. (2015) proposed a possible approach for outsourcing process mining, which is capable of preventing the confidentiality of data when operating cross-organizational mining by the encryption of strings and numerical attributes. Nonetheless, this approach was implemented in ProM and most of the analysis plugins do not involve numerical data. So, each plugin needs to be appropriately modified and adopt a full encryption of numerical values. Concerning the other presented approaches, the confidentiality of data has not been considered. Thus, besides comparative approaches, concentrating the focus toward privacy and security problems related to cross organisational cooperation is necessary. Table 1.6 summarises the techniques applied to handle cross-organizational mining.

Table 1.6. Techniques applied for cross-organizational mining.

Paper Ref.	**Used methodology**	**Outcome**
Buijs *et al.* (2012b)	Generic approach	Compare collections of process models and their events logs across organisations
Buijs *et al.* (2013b)	ETM genetic algorithm based approach	Discover configurable process model based on a collection of event logs
Zeng *et al.* (2013)	Coordination patterns and RM_WF_Net based approach	Discover a model for a cross-organizational workflow
Buijs *et al.* (2014b)	Alignment based method	Compare procedure of similar processes across organisations
Schunselaar *et al.* (2015a)	Implementation of YAWL into the cloud	Support different variants of the same process across non-competitive organisations
Sebu *et al.* (2015)	Process mining algorithms and graph comparison methods based approach	Determine the most suitable organisations for profitable collaborations
Yilmaz *et al.* (2015)	Unsupervised learning based cross-organizational mining	**G**enerating recommendations using cross-organizational process mining for process

	framework	performance improvement
Burattin *et al.* (2015)	AES and Paillier cryptography based approach	Prevent the confidentiality of data when operating cross-organizational mining
Aksu *et al.* (2016)	Generic Cross-Organizational Process Mining Framework	Accurately compare organisations based on their event logs and semantic and organisational context

1.4.8 Providing operational support

Nakatumba *et al.* (2012b) suggested a concrete implementation of operational support meta-model using the workflow system Declare and the process mining framework ProM. The meta-model includes four types of queries of increasing complexity and power: simple queries, compare queries, predict queries, and recommend queries. However, research papers that use process mining algorithms under this framework for dealing with operational support are needed.

Bose and Van der Aalst (2013) explored the feasibility of a global approach for signatures discovery that can be used to explain or predict the classes of visible and invisible traces. The discovered signature patterns allow the distinction between various classes of behavioor.

Conforti *et al.* (2013a) offered a method that predicts process risks by applying decision trees to the logs of previous process executions taking into account process data, used resources, task durations, and contextual information. The proposed method helps the process participants to make risk-informed decisions. However, evaluation of the approach is still required.

Leoni and Van der Aalst (2014) explored an approach that forecasts the remaining processing time, and recommend activities to reduce risks. They offered also a general approach to correlate process characteristics.

Hompes *et al.* (2015) explored a new trace clustering approach based on the Markov cluster (MCL) algorithm with the ability to detect changes of a process according to the selected perspectives. Both positively and negatively deviating cases can be used to enhance the process and/or to prevent the undesirable behaviour from occurring in the future.

Leoni et al. (2016a) proposed a generic framework for predicting dynamic behaviour from event logs. Besides, it is capable of correlating and clustering dynamic behaviour. The framework allows the prediction of the executor of a certain activity, the remaining time to the end of the process instance, the next activities to work on, and the outcome of the executions of process instances.

All of the cited papers reported on the successful application of process mining for operational support, especially in terms of prediction and recommendation. However, the main challenge of providing operational support by process mining techniques has not been addressed. When applying process mining methods to detection, prediction, and recommendation, the problem of handling computing power and data quality issues arise but have not been well considered. Therefore, next researches are called to focus on the mentioned problems besides the applicability of process mining techniques for operational support. Table 1.7 summarised the reported publications.

Table 1.7. Main techniques for providing operational support

Paper Ref.	**Used methodology**	**Outcome**
Nakatumba *et al.* (2012b)	A meta-model for operational support	Provide statistics about the current execution, compare the current execution, predict the outcome of the current execution, and recommend what to do
Bose *et al.* (2013)	A comprehensive framework for discovering signatures	Help to explain or predict the class of seen or unseen traces
Conforti *et al.* (2013)	Decision trees based approach	Aid process participants to make risk-informed decisions
Leoni *et al.* (2014)	Feature prediction framework	Forecast remaining processing time, and recommend activities to reduce risks
Hompes *et al.* (2015a)	Clustering approach based on Markov cluster algorithm	Detect process changes and prevent unwanted behaviour from occurring in the future
Leoni et al. (2016a)	Generic framework for predicting dynamic behaviour	Predict the executor of a certain activity, the remaining time to the end of the case, the next activities to work on, and the outcome of the executions of cases

1.4.9 Combining process mining with other types of analysis

The challenge of combining process mining techniques with other types of analysis has received significant focus from researchers by providing frameworks of mapping successfully process mining with simulation, big data, data mining, analytic workflow systems, visual analytics, patterns mining, and Indoor location systems. We mined in Table 1.8 these combinations from research papers addressing the present challenge. Amalgamating process mining approaches with such techniques engenders many benefits. For instance, combining process mining methods with *data mining techniques* such as association rule learning offers solutions for fraud detection (Sarno *et al*., 2015), with *visual analytics* produces innovative process-centric visualizations, such as "process movies" proposed by Leoni, M. *et al.* (2015a), with *simulation* leads to better understanding, modelling, and improving real-life business processes (Nakatumba *et al.,* 2012a), with *big data* characteristics generates all the applicable benefits of Big Data in business process mining: Adequacy for discovery, adequacy for prediction, visibility, flexibility for efficiency, and flexibility for conformance (Omair and Emam, 2015), and with *analytic workflow systems* helps design, compose, execute, archive, and share workflows that constitutes the image of some type of analysis (Bolt *et al.*, 2015).

Table 1.8. Combinations achieved between process mining and other types of analysis

Reference of papers	**Type of combinations**
Nakatumba *et al.* (2012a)	Process mining ↔ Simulation
Aguirre *et al.* (2013)	Process mining ↔ Simulation ↔ Data mining ↔ Tools of the understanding phase of the BPTA
Leoni, M. *et al.* (2014)	Process mining ↔ Data mining (Decision Tree)
Leemans, M. *et al*. (2014)	Process mining ↔ Pattern mining
Bolt *et al.* (2015)	Process mining ↔ Analytic workflow systems
Leoni, M. *et al*. (2015a)	Process mining ↔ Visual analytics
Sarno *et al*. (2015)	Process mining ↔ Data mining (Association Rule Learning)
Omair and Emam (2015)	Process mining ↔ Big Data
Evermann *et al.* (2015)	Process mining ↔ Data mining (Clustering)
Fernandez -Llatas *et al.*	Process mining ↔ Indoor location systems

(2015)	
Pileggi *et al.* (2015)	Process mining ↔ Simulation
Leoni *et al.* (2016a)	Process mining ↔ Data mining (Clustering + Regression Tree)
Leoni *et al.* (2016b)	Process mining ↔ Visual analytics

1.4.10 Improving usability for non-experts

Table 1.9 provides researchers with the state of art of what has been done for improving the usability of process mining techniques for non-experts. This will help them identify where they can orient their research focus regarding this challenge. Many process mining tools and frameworks have been developed, and their usability has been improved for the sake of end users who are not necessarily experts in process mining. However, the current tools need to be enhanced to enable researchers and business users to use process mining tools for different purposes. For instance, sophisticated tools for process discovery that allows users to modify the variables that impact all perspectives of a process need to be developed.

Table 1.9. Recently developed tools and approaches for improving usability for non-experts

Reference of papers	**Tool/Approach**	**Purpose/function**
Adriansyah *et al.* (2012)	YAWL based event log replayer tool	Allow users to replay, analyse, and visualise a variety of performance metrics of an AS-IS or TO-BE process models
Gunther *et al.* (2012)	Disco tool	Allow users to easily and quickly generate visual and actionable insight about processes from raw data
Ramezani *et al.* (2013)	Framework for creating and understanding formal compliance requirements	Allow compliance specification for end users without extensive knowledge
Shershakov (2013)	DPMine tool	Enable users to build a model of multistage process mining from individual processing units connected to each other in a processing graph

Mans *et al.* (2014)	RapidMiner 5	Aid users to define and execute analysis of workflows connected to the process mining framework ProM 6
Kalenkova *et al.* (2014a)	BPMN support in ProM	Support ProM users to bridge the gap between formal models (such as Petri nets…) and process models used by practitioners
Van Zelst *et al.* (2014)	Single-Node stream extension of ProM	Provide users with the possibility to handle effectively data stream (Real-time data)
Leemans, S. *et al.* (2015b)	Inductive visual Miner (IvM) (process exploration tool)	Bridge the gap between academic and commercial process exploration tools by considering zoomability, evaluation, semantics, and speed
Shugurov and Mitsyuk (2015)	Iskra: tool for process enhancement (Decomposed model repair)	Allow flexible repair configuration. The tool can be easily used by both researchers and developers within ProM community
Schunselaar *et al.* (2015b)	Automatic framework to an easy configuration of configurable process model	Allow end users to configure easily a configurable process model
Leoni *et al.* (2016a)	Generic environment for predicting, correlating and clustering dynamic behaviour from event logs	Allow process analysts without a solid technical background to conduct quickly multiple process centric analyses
Lu *et al.* (2016a)	Log to Model Explorer tool	Aid users to interactively and iteratively explore and preprocess a log, and discover suitable models from it
Leoni *et al.* (2016b)	Log On Map Replayer tool indexed in ProM	Enable process analysts to dynamically visualize the replay of the behaviour of executed process instances as recorded in the event log

1.4.11 Improving understandability for non-experts

Maggi *et al.* (2012b) offered a method to simply determine comprehensible Declare models (which is composed of temporal constraints) using Apriori algorithm. The distinctive feature of their approach is that it produces only the candidate constraints. Moreover, they used association rule mining based criteria to assess the pertinence of a discovered constraint. This approach generates understandable process models.

Zhao *et al.* (2014) incorporate genetic algorithm with the role complexity of process models and presented the role-based process mining technique to discover the simplified process model. Based on a new role metric of role based process complexity, the approach results in process models easy to understand.

Fuzzy models are well known and easy to understand compared to other existing models. Therefore, Shershakov (2015) introduced a new approach for mining fuzzy models which is based on relational database management technique that provides various data views for different types of analysis.

Leoni *et al* (2016b) developed an approach named *Log On Map Replayer* implemented in ProM that is capable of visualizing the replay process histories as recorded in the event log in a dynamic way. This framework has been developed such that the user can understand easily what has happened with executed processes and can draw meaningful conclusions regarding the behaviours and/or performance of their processes.

It seems that improving understandability for non-experts didn't receive a considerable focus as few works have been done in this area. Although fuzzy models, comprehensible declare models, and role based process models by genetic algorithm are easy to understand, if the four quality dimensions are not specified in the result, users might end up by drawing wrongs conclusions and thus making wrong decisions. Therefore, further researches that produce understandable process model with the value of the four quality criteria are required.

The aim of this section is to increase the maturity of the field of process mining by providing researchers with the state-of-the-art of process mining challenges.

The first challenge of finding, merging and cleaning event logs has received a significant focus from research community by dealing with event log stored in various data sources, event data that had been executed in a certain context, event data that is object centric than process centric, incompleteness, noise (infrequent behavior), and event data characterized with different levels of granularity.

To deal with complex event logs a considerable number of researches agreed that process mining decomposition is the best solution. However, the approaches used for decomposition were not consistent among studies: some used the divide-and-conquer approach, some used the Single-Entry Single-Exit technique, some used the notion of process cubes, and others have based their decomposition on clustering. Therefore, a benchmark of these decomposition strategies is required for selecting the appropriate technique. Moreover, each study for process mining decomposition has some limitations and need to be investigated in further works. For instance, the conflicting quality dimensions have not been fully considered in their studies.

The challenge of combining process mining techniques with other types of analysis is receiving significant focus from researchers by providing frameworks of mapping successfully process mining with simulation, big data, data mining, analytic workflow systems, visual analytics, patterns mining, etc.

Regarding improving usability for non-experts, although many process mining tools and frameworks have been developed for users who are not necessary experts in process mining, the current tools need to be enhanced to enable researchers and business users to use process mining tools for different purposes.

Although a substantial number of solutions for dealing with concept drift have been proposed, only changes from the control-flow perspective were considered, whereas data and resource perspectives are equitably important.

To handle the challenge of cross-organizational mining several analytical techniques have been developed in many papers and focused on commonality and collaboration between organisations. However, most of the publications didn’t consider the big issue of confidentiality of event logs and processes.

Concerning the challenge of balancing the conflicting quality criteria, it has been proven that some process discovery algorithms are able to balance the four quality dimensions producing representation such as process trees. However, not all existing process discovery techniques can actually produce process trees. The balance framework should not be restricted to algorithms discovering one specific notation.

Regarding the challenge of providing operational support, all published papers demonstrated the successful application of process mining for detection, prediction, and recommendation. Nevertheless, when applying process mining methods to this online setting, the problem of handling computing power and data quality issues arise and have not been considered yet.

To create representative benchmarks, different frameworks have been developed. But, each framework has limitations. A good benchmark platform should not take reference models as input since they are not usually available. Moreover, the framework should not be time-consuming.

Whereas the focus; to improve the representational bias used for process discovery; should be toward the implicit search space implied by the representational bias, most publications focused on understandability, correctness, and quality of the representation and also on transformations techniques that convert control flow modelling representations to the desired language visualisation. In our opinion, since the characteristics of event logs strongly influence representational bias, if frameworks that determine the features of event logs without using control-flow algorithm exist, selecting the right representation bias would be very clear.

Very few works were performed to improve the understandability for the non-expert. Further researches that produce understandable process model by specifying the quality metrics are required.

This paper underlined limitations of the reviewed publications regarding process mining challenges. The highlighted limitations tend to be a starting point for other researches in the field of process mining, specifically concerning process mining challenges.

1.5. Process discovery techniques limitations overview

One of the major process discovery techniques is the $\alpha -$algorithm (Van der Aalst et al., 2004). This algorithm produces a workflow net based on the causal relationships observed between tasks, but without short loops, invisible tasks, non-free choice constructs, duplicate tasks, or noise. Therefore, several extensions have been presented so far to tackle these limitations. The Alpha + algorithm was first introduced to extend the Alpha algorithm to short loop mining (De Medeiros et al., 2004), while Alpha ++ was developed to extend the Alpha algorithm to non-free choice constructs (Wen et al., 2007a), Alpha # was developed to mine invisible tasks (Wen, 2007b; 2010), and Alpha $ was developed to handle invisible tasks involved in non-free choice constructs (Guo et al., 2015). However, none of these extensions can detect all types of characteristics that can exist in an event log. HeuristicsMiner (Weijters et al., 2006) derives ordering relations on the basis of their frequencies and returns a net. The HeuristicsMiner algorithm is robust for noisy logs and invisible tasks, but cannot handle duplicate tasks and non-free choice constructs. The Inductive Miner returns a process tree and guarantees sound models that other algorithms cannot guarantee (Leemans et al., 2013). It is robust for noisy logs and capable of mining invisible tasks, but cannot discover duplicate tasks or non-free choice constructs. The Region-based algorithm can mine non-free choice constructs, but cannot handle invisible tasks and duplicate tasks (Bergenthum et al., 2007). The Genetic Mining algorithm is the only existing algorithm that can deal with most of the common constructs and noisy logs (Van der Aalst et al., 2005). Nevertheless, it requires many parameters and does not provide the correct output in a restricted time. The aforementioned characteristics will be explained in Chapter 4.

1.6. Contributions and Structure of the Thesis

The first contribution of this thesis is to provide the state-of the art of the most important challenges of process mining by conducting a critical and comprehensive review on 105 publications identified dealing with process mining challenges and published from the beginning of 2012 till the beginning of 2017.

The contribution of the present work is to demonstrate the importance of process mining in the practical world specifically in the industrial field by applying process mining techniques to a real customer order fulfillment process of a heavy manufacturing industry to analyze the behavior and performance of this process.

The critical review we conducted on the most important challenges of process mining specified in the Process Mining Manifesto shows that the challenge "*creating a representative benchmark*" or "*a recommendation framework of process discovery algorithms*" is one of the important challenges that received less focus from researcher community among the other challenges. A lot of process discovery techniques are available. However, users and businesses still cannot choose or decide the appropriate mining algorithm for their business processes. Each algorithm has a specific limitation regarding mining the constructs of a process model. Therefore, the third contribution of this thesis is the development of a framework that recommends the process discovery algorithm suitable for a given process.

Each process discovery technique has a specific restriction in terms of mining certain constructs of a process model such as short loops, invisible tasks, duplicate tasks and non-free choice constructs. There is no algorithm which is capable of discovering the aforementioned characteristics in a restricted time if all of them are present in the event log. The fourth contribution of the present study is the development of a new process discovery technique capable of handling the aforementioned constructs.

The first part of this thesis provides an introduction to limitations of process-aware systems, process mining, and a critical review on the most important challenges of process mining by providing the state-of-the art, and finally an overview of process discovery techniques limitations. The second Chapter presents an industrial application of process mining to show its importance in the practical world. Furthermore, preliminaries are provided in Chapter 3. The standard and complex constructs of a process model are introduced in Chapter 4. The first major part of this thesis is presented in Chapter 5 which introduces our recommendation framework for process discovery techniques and an evaluation of the framework based on artificial and real-life datasets. The second major part of the present study which is about developing a new process discovery approach capable of dealing with the standard and complex constructs of a process model is presented in Chapter 6. This process discovery technique is evaluated using both artificial and real-life data in Chapter 7. Last Chapter provides limitations and future works of the two introduced framework. Finally, the thesis is concluded.

Chapter 2 INDUSTRIAL APPLICATION OF PROCESS MINING

Process mining has been widely applied in multiple domains through several cases studies. The most commonly tackled field is the healthcare as 74 case studies applied to this domain have been identified according to (Rojas et al., 2016). This considerable number illustrates the huge interest of hospitals in process mining. An example of healthcare related case study can be found in (Cho et al., 2015). Although process mining has been applied successfully to other domains such as education (Vazquez Barreiros et al., 2014), IT services (Vazquez Barreiros et al., 2016), industry (Park et al., 2015), etc., it has not been applied as much as the healthcare area. Therefore, with the aim to contribute to increasing the applicability of process mining in other fields, specifically in the industry field, we present an application of process mining in a real customer order fulfillment process of ship parts manufacturing company. This process is characterized by a very long lead time as it takes 34 weeks in average to fulfill a customer order. This company adopts a customized BPM system to manage and control their processes. The datasets used in this case study is extracted from this system and consists of a total of 1653 different customer orders and 63812 events generated between February 2012 and March 2016. Although this organization uses a BPM system, they have no clear idea about what is really happening in the customer order fulfillment process and how this process is handled. The first contribution of this case study is to identify, understand, and analyze the as-is process of the customer order fulfillment, determine deviations as well as what makes this process spending too much time based on the three types of process mining (i.e., process discovery, conformance checking and performance analysis). By identifying the factors that affect negatively the performance of the whole process, the company can tailor their focus on these factors to improve and optimize this process. The second contribution is to demonstrate the applicability of process mining technique in the customer order fulfillment process of a ship and naval manufacturing company, while the third contribution is to increase the number of applications of process mining in the industry field especially in manufacturing to provide businesses with the importance of process mining in understanding the as-is process and encourage them to refer to process mining to optimize their processes.

2.1. Literature review on process mining case studies

Real-life applications are indispensable to demonstrate the utility and the importance of process mining in the practical word. The more we apply process mining to various fields via multiple case studies, the more we can understand the potential of process mining to handle real problems and the more we can identify the process mining scopes that can or need to be improved to make process mining techniques suitable to the real practice. The industry field is one of the main areas that can largely benefit from process mining techniques. However, the number of existing case studies on industrial applications remains insufficient.

This chapter listed eight relevant case studies in Table 2.1 which reported on industrial applications of process mining. The first industrial application was described in (Van der Aalst et al., 2007). This case study was applied on a process of handling invoices in a provincial office responsible for the construction and maintenance of the road and water infrastructure. The goal of this study was to prove the applicability of process mining according to the three perspectives of process discovery: control-flow, organizational and case perspectives. In our case study, an application of process mining techniques to customer order fulfillment process in a ship and naval parts manufacturing company is presented, and not only the process discovery type is applied but also the conformance checking and process enhancement. Based on the data collected from the BPM system of the company only control-flow perspective can be applied regarding the process discovery type. The second industrial application was conducted by (Rozinat et al, 2009) on a test process of a wafer scanner manufacturing company. The main focus of this case study was on the conformance verification type of process mining. In (Goedertier et al., 2011), a process mining case study was described on a customer invoice handling process of a telecom industry. Unlike our process mining application, the purpose of this study was an empirical evaluation of three process discovery algorithms: Genetic Miner, AGNEs and HeuristicsMiner. Another case study in the telecom industry was described in (Era et al., 2015). In this application, process mining is applied to customer fulfillment process. The main focus of this study was identifying bottlenecks in the process. The name of the process analyzed in this case study and the process analyzed in our study is similar. However, the type of the industry, the type of the algorithm applied to discover the as-is process and the approaches used to analyze the customer fulfillment process are distinct. Moreover, the process under our case study is more complicated. In (Taylor et al. 2012), a case study was performed on a set of multinational enterprises with the aim to underline the issues related to process data encountered while conducting process mining analysis. The author stressed on the fact that most of the literature studies don't describe in detail how the process data is handled. In our case study, the pre-processing of the collected data is explained in detail. In (Aruna et al., 2012), the applicability of process mining was described through a case study in a corrugated boxes manufacturing process according to the organizational perspective. In (Son et al., 2014), the authors conducted a case study on the production process of an Electro-Mechanic manufacturing company. In this case study, process mining techniques were applied according to the control-flow and machine perspectives with the purpose of proving the applicability of process mining techniques in production processes in manufacturing company. Finally, in (Park et al., 2015), the authors presented a method to analyze a case study manufacturing processes in make-to-order production with process mining. The major focus of the authors was the performance analysis including workload and delay analysis.

These case studies show the applicability of process mining in the industrial sector based on different types and perspectives of process mining techniques. However, this is not sufficient. More real-life case studies are required to prove the important role that process mining plays in analyzing the as-is process and thus encourage businesses to adopt process mining to optimize their processes.

Table 2.1. An overview of industrial applications of process mining in the practical world

Author	**Industry type**	**Analyzed process**	**Process mining types**				
			Type 1			**Type 2**	**Type 3**
			A	**B**	**C**		
Van der *Aalst et al.*, 2007	Provincial office of road and water infrastructure construction and maintenance	Invoice handling	√	√	√		
Rozinat *et al.*, 2009	Wafer Scanner manufacturing	Test process				√	
Goedertier *et al.*, 2011	Telecom industry	Invoice handling	√				
Taylor *et al.*, 2012	Multinational corporation	-	√				√
Aruna *et al.*, 2012	Corrugated Boxes manufacturing	Production		√			

Son *et al.*, 2014	Electro Mechanic Manufacturing	Production	√	√		√	√
Era *et al.* 2015	Telecom industry	Customer fulfillment	√				√
Park *et al.*, 2015	Shipbuilding manufacturing	Production					√

Type 1: Process discovery, Type 2: Conformance verification, Type 3: Performance Analysis, A: Control-flow perspective, B: Organizational perspective and C: Case perspective.

2.2. Case study

With the aim to demonstrate and boost the usefulness of process mining analysis in the real word, in this study, we describe a real-life case study in a ship and naval parts manufacturing company in Korea that is producing steel structures, engine tools, cell guides and peripheral apparatuses for shipbuilding and marine industries.

In order to execute, monitor and control its operational business processes, the company adopts a customized BPM system named Shipbuilding Processing Plan Management (SPPM) System. This system provides a lot of information related to process execution. However, since BPM systems are completely disconnected from actual data as they are based on the idealized model of reality, the company has no clear idea of what is really happening in the customer order fulfillment process and how this process is handled. Thus in this research, we applied process mining techniques to analyze the customer order fulfillment process, identify the actual as-is process (what is actually happening) and compare it with the supposed as-is process (what the company think is happening) and finally determine the parts that affect the performance of the whole process.

2.2.1. Case Description

The case study described in this paper is the analysis of the customer order fulfillment process in a heavy manufacturing industry. Customer order fulfillment process refers to the entire process from the point a customer makes an order to the delivery of the product to this customer. This process is handled on 7 successive phases: (1) order handling, (2) required material purchasing, (3) production, (4) quality inspection, (5) post-processing, (6) packing and delivery, and (7) customer order fulfillment verification. In the first phase, once the company receives an order from a particular customer to manufacture its desired products, purchase department defines a construction ledger to determine the amount of expenses associated with the fulfillment of the order received from the customer. A construction ledger is a record that includes details about the contracts being concluded between the customer and the manufacturing company including all charges associated with a construction (manufacturing) project. The order is then approved by the business department and then by the production department. As soon as the order is approved, a verification of the order approval is executed and then the order approval is finalized. In the second phase, required materials for manufacturing the customer's need are purchased and stored in the warehouse as they arrive. In the third phase, the production process is conducted. As soon as the production line finishes, two types of quality inspection are performed in the fourth phase to validate the quality requirement: self-inspection and then external inspection. After this, a post-processing is carried out by performing painting and plating operations. The final product is then packed and delivered to the customer in the sixth phase. Finally, in the last phase, the completion of the order is approved and then the approval is verified by production and business departments. If the manufactured products satisfy the customer, the order completion is finalized and the process is closed. The overall process of customer order fulfillment is described in Figure 2.1.

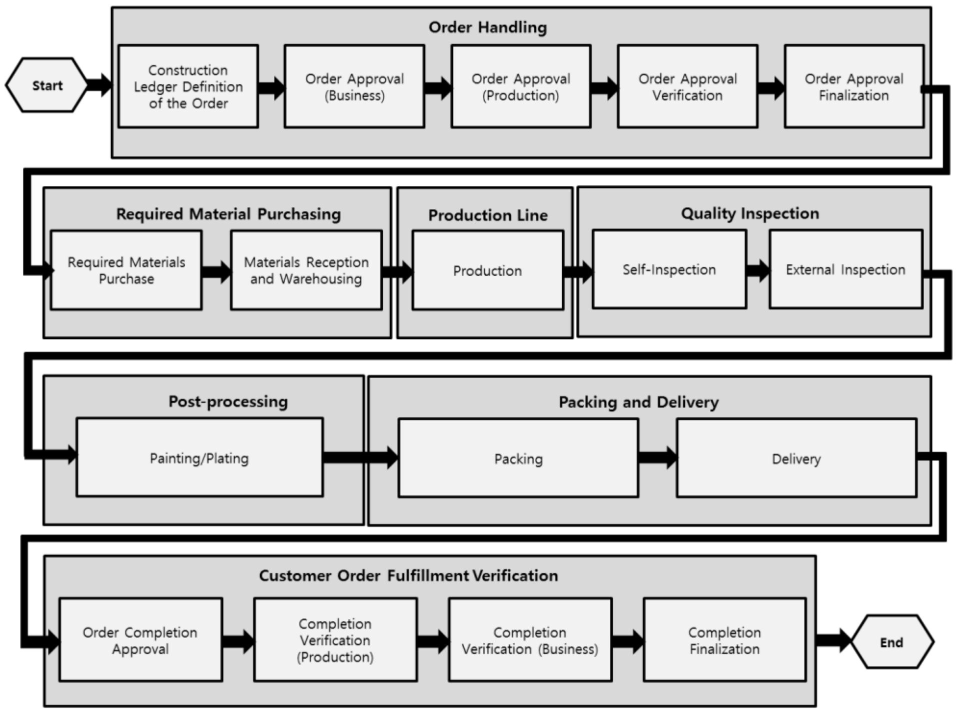
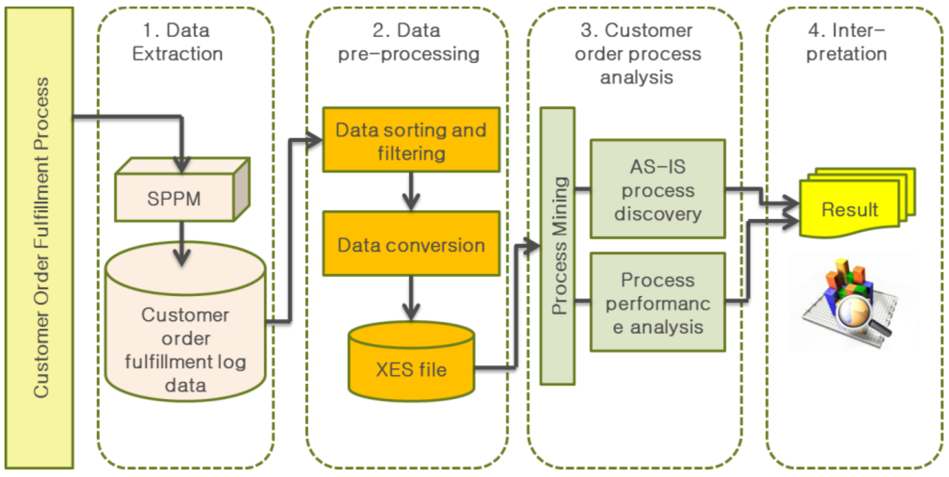

Order Handling
Start
Construction Ledger Definition of the Order
Order Approval (Business)
Order Approval (Production)
Order Approval Verification
Order Approval Finalization
Required Material Purchasing
Required Materials Purchase
Materials Reception and Warehousing
Production Line
Production
Quality Inspection
Self-Inspection
External Inspection
Post-processing
Painting/Plating
Packing and Delivery
Packing
Delivery
Customer Order Fulfillment Verification
Order Completion Approval
Completion Verification (Production)
Completion Verification (Business)
Completion Finalization
End

Figure 2.1. Customer order fulfillment standard process description

2.2.2. Application Methodology

The framework of the Analysis consists of four steps: data extraction, data pre-processing, customer order fulfillment process analysis, and interpretation of results as shown in Figure 2.2.



Customer Order Fulfillment Process
1. Data Extraction
SPPM
Customer order fulfillment log data
2. Data pre−processing
Data sorting and filtering
Data conversion
XES file
3. Customer order process analysis
Process Mining
AS−IS process discovery
Process performance analysis
4. Inter−pretation
Result

Figure 2.2. Customer order fulfillment process analysis framework

2.2.3. Event log construction

The starting point of any process mining analysis is the preparation of the available process data. This step consists of the identification of activities and their attributes such as Case ID and timestamps related to the customer order fulfillment process to build an event log and then the conversion of the constructed event log into a standard log format supported by process mining tools. In this phase, the process related data from February 2012 to March 2016 are extracted from the SPPM system of the company. Because the obtained data contained a lot of unnecessary information, we sorted the collected data based on case id, activity, and timestamps. The case id in this process is a customer order. The activities and their attributes are shown in Table 2.2. But before transforming the log into the standard event log format, i.e., XES (Van Dongen et al., 2015) using ProM tool (Van Dongen et al. 2005), we performed the following pre-processing steps:

- Identification of start timestamps and end timestamp of all activities in each trace: Raw data contained information about tasks executed in order and associated with a case ID and with the status of the execution of these tasks. The status of each activity was specified by one of the following flags: (S, F), (S, X), and (S, R). ‘S’ shows that the execution of a particular task is started; ‘F’ indicates that the execution of the task is finished; ‘X’ refers to the situation when the completion of a task is forced, and finally ‘R’ shows that the outcome of the task is returned to be executed or recertified by the previous task. To each of these five statuses, a timestamp is assigned. However, in process mining analysis only start timestamps (i.e., in this case start timestamps is the timestamp associated with ‘S’) and end timestamps (i.e., the timestamp associated with ‘F’) are needed. All activities have the start timestamp associated with them but not all activities own the end timestamp (i.e., ‘F’, ‘X’, and ‘R’). But if we look at the definition of the attributes ‘X’ and ‘R’ we realize that the timestamps associated with ‘X’ and ‘R’ are actually end timestamps. For instance, a task with an X status refers to a forced termination. This implicitly indicates the completion of the task. Accordingly, we have converted ‘X’ and ‘R’ flags into ‘F’ flag such that to all tasks, the pair (S, F) is assigned. A captured screen of the result obtained in this pre-processing is depicted in Figure 2.3(a).

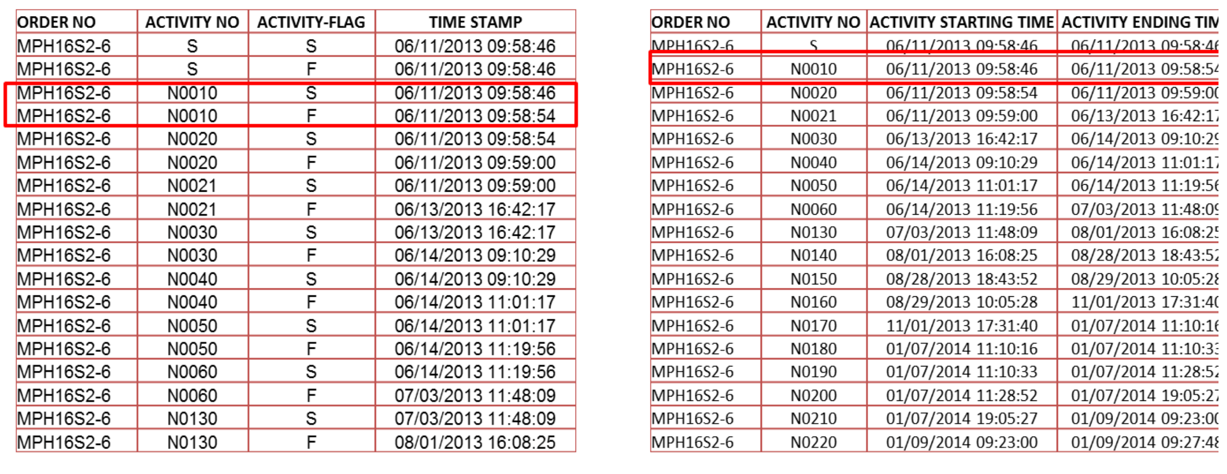
- Transformation of the format of the table containing the sorted events attributes: After converting the ‘X’ and ‘R’ attribute into ‘F’ status, the status of each activity is represented by ‘S’ status (start timestamp) and ‘F’ status (end timestamp). However, the obtained data were sorted in a way such that each task appears two times in two successive rows. The start timestamps (and ‘S’ status) are indicated in the first raw and the end timestamps (and ‘F’ status) are indicated in the second row of each activity. In order to be able to convert the event log into XES format, we have transformed the table shown in Figure 2.3(a) into the event log shown in Figure 2.3(b) such that the start timestamp and end timestamp of each activity are depicted in two columns rather than two successive rows.

- Construction of a structured event log: in the obtained data, the End activity of all process instances contained only the end timestamp attribute, while N0060 activity contained only the start timestamp attribute. In order to obtain a structured event log, we set the start timestamp of End activity of each process instance equal to the end timestamp. Furthermore, we found that the end timestamp of N0060 activity was actually recorded in the execution file of a different process, i.e., a sub-process connected to the customer order fulfillment process. Hence, the data related to the end timestamps of N0060 activity is collected and included in the constructed event log.

After the pre-processing steps are conducted, the constructed event log is converted into the standard format of event logs, i.e., XES, forming the input of our process mining analysis.

Table 2.2. The attribute of activities of customer order fulfillment process

Activity	Attribute
Start (Receive a customer order)	S
Order (Construction Ledger)	N0010
Order (Business) Approval	N0020
Order (production) Approval	N0021
Order Approval Verification	N0030
Order Approval Finalization	N0040
Material Purchase Request	N0050
Waiting for Requested Material Purchase	N0060
Production	N0130
Self-inspection	N0140
External inspection	N0150
Post-processing (painting/plating)	N0160
Packing	N0170
Delivery	N0180
Completion Approval	N0190
Completion (production) verification	N0200
Completion (business) verification	N0210
Completion finalization	N0220
End (order received by the customer)	E



ORDER NO	ACTIVITY NO	ACTIVITY-FLAG	TIME STAMP
MPH16S2-6	S	S	06/11/2013 09:58:46
MPH16S2-6	S	F	06/11/2013 09:58:46
MPH16S2-6	N0010	S	06/11/2013 09:58:46
MPH16S2-6	N0010	F	06/11/2013 09:58:54
MPH16S2-6	N0020	S	06/11/2013 09:58:54
MPH16S2-6	N0020	F	06/11/2013 09:59:00
MPH16S2-6	N0021	S	06/11/2013 09:59:00
MPH16S2-6	N0021	F	06/13/2013 16:42:17
MPH16S2-6	N0030	S	06/13/2013 16:42:17
MPH16S2-6	N0030	F	06/14/2013 09:10:29
MPH16S2-6	N0040	S	06/14/2013 09:10:29
MPH16S2-6	N0040	F	06/14/2013 11:01:17
MPH16S2-6	N0050	S	06/14/2013 11:01:17
MPH16S2-6	N0050	F	06/14/2013 11:19:56
MPH16S2-6	N0060	S	06/14/2013 11:19:56
MPH16S2-6	N0060	F	07/03/2013 11:48:09
MPH16S2-6	N0130	S	07/03/2013 11:48:09
MPH16S2-6	N0130	F	08/01/2013 16:08:25

(a)

ORDER NO	ACTIVITY NO	ACTIVITY STARTING TIME	ACTIVITY ENDING TIM
MPH16S2-6	S	06/11/2013 09:58:46	06/11/2013 09:58:4
MPH16S2-6	N0010	06/11/2013 09:58:46	06/11/2013 09:58:5
MPH16S2-6	N0020	06/11/2013 09:58:54	06/11/2013 09:59:0
MPH16S2-6	N0021	06/11/2013 09:59:00	06/13/2013 16:42:1
MPH16S2-6	N0030	06/13/2013 16:42:17	06/14/2013 09:10:2
MPH16S2-6	N0040	06/14/2013 09:10:29	06/14/2013 11:01:1
MPH16S2-6	N0050	06/14/2013 11:01:17	06/14/2013 11:19:5
MPH16S2-6	N0060	06/14/2013 11:19:56	07/03/2013 11:48:0
MPH16S2-6	N0130	07/03/2013 11:48:09	08/01/2013 16:08:2
MPH16S2-6	N0140	08/01/2013 16:08:25	08/28/2013 18:43:5
MPH16S2-6	N0150	08/28/2013 18:43:52	08/29/2013 10:05:2
MPH16S2-6	N0160	08/29/2013 10:05:28	11/01/2013 17:31:4
MPH16S2-6	N0170	11/01/2013 17:31:40	01/07/2014 11:10:1
MPH16S2-6	N0180	01/07/2014 11:10:16	01/07/2014 11:10:3
MPH16S2-6	N0190	01/07/2014 11:10:33	01/07/2014 11:28:5
MPH16S2-6	N0200	01/07/2014 11:28:52	01/07/2014 19:05:2
MPH16S2-6	N0210	01/07/2014 19:05:27	01/09/2014 09:23:0
MPH16S2-6	N0220	01/09/2014 09:23:00	01/09/2014 09:27:4

(b)

Figure 2.3. Event log pre-processing, (a) before and (b) after transforming the format of the table containing the sorted events attributes

2.2.4. Event log filtering

The data retrieved for this study is from February 2012 to March 2016. Therefore, certainly there exist cases in the extracted data that may have actually started before February 2012 and also cases that have not finished yet and might be finished after March 2016. This refers to the situation of incomplete cases. As process mining techniques can only operate on complete process instances, it is necessary to filter the incomplete events.

To process the filtering, we imported the event log in ProM tool and visualize it. We found that it consists of 19 different activities, 1695 cases with 64988 events, 2 start activities (S and N0010), and 7 end activities (E, N0130, N0021, N0060, N0020, N0010, and N0180). After filtering the event log by running the action "Filter Log using Simple Heuristics", we found that it contains 19 different activities, 1653 cases which is 97% of the original event log, one start activity (S), and one end activity (E). The result of log filtering is depicted in Table 2.3.

Table 2.3. Event log information before and after filtering

Event log characteristics	**Before Filtering**	**After Filtering**
Total number of cases	1695	1653
Total number of events	64988	63812
Total number of activities	19	19
Total number of start activities	2	1
Total number of end activities	7	1

2.2.5. Customer Order Fulfillment Process Discovery

Process discovery techniques take an event log as input and automatically construct a process model. The main idea behind the process discovery is to find the as-is process model or in other words to find what is really happening in the company. Using process discovery, many business questions can be answered. In the first place, process discovery allows businesses to confirm or correct their idea regarding what they think is happening in the business process. Furthermore, the presence of a process model that exactly represents the real behavior of a business process enables forward analysis such as performance analysis, bottleneck analysis, or identification of path that are not in conformance with business rules (Goedertier et al., 2011). Until now, plenty of process discovery algorithms have been developed such as α_algorithms (van der Aalst et al., 2003), genetic miner (van der Aalst et al., 2005), heuristic miner (Weijters et al., 2006), inductive miner (Leemans et al., 2013), fuzzy miner (Günther et al., 2007), etc. In this study, we used the fuzzy miner. This algorithm allows discovering process models from very large event logs, in less than 1 second and easy to understand.

We applied the fuzzy miner using Disco tool (Gunther et al., 2012) on the filtered event log. The discovered model of the customer order fulfillment process is illustrated in Figure 2.4. In the resulting derived model, all cases start with the task 'S' and end with the task 'E'. The numbers, the thickness of transitions and the coloring depict how frequently each activity and path has been executed.

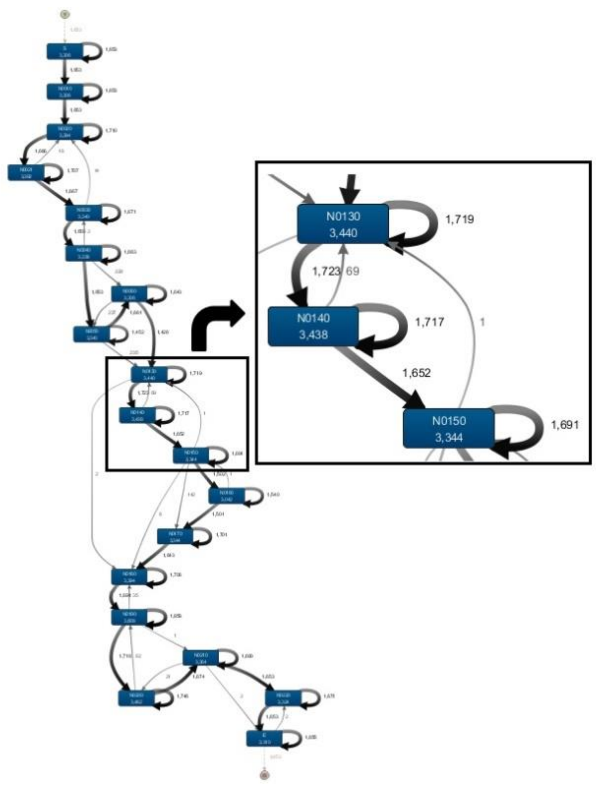


Figure 2.4. The discovered process models using Disco tool from the filtered event log

2.2.6. Compliance verification

Compliance verification in process mining is a set of methods that allow comparing a business process model with the event log of the same process model. The goal of compliance verification is to investigate whether the actual execution of a business process, i.e., the event log and the modeled business process are in compliance with each other. To do so, the fitness metric; which measures how much the process model can reproduce the traces recorded in the event log; is measured. One way to find the fitness is to reply the log in Petri net (Van der Aalst et al., 1998). Log replay is operated such that if there are missing tokens to fire the transitions in question, they are created artificially to continue the reply. Token-based fitness metric f computes the amount of missing and remaining tokens during the log replay. If the log could be replayed correctly the value of f is equal to 1.

Compliance verification allows identifying differences and deviations between the modeled behavior (the standard model) and the observed behavior (the event log). Two types of differences can be detected: (i) a behavior observed in the event log, while it is not allowed by the model, and (ii) a behavior allowed in the model but never captured in the log. These discrepancies could be determined by one of the following techniques: replay (Rozinat et al., 2008), trace alignment (Adriansyah, 2014), and behavioral alignment (Goedertier et al., 2009). In this study, trace alignment method is selected because it can handle complex control-flows that contain invisible tasks in a Petri net (Van der Aalst et al., 1998). An alignment between a recorded process execution (i.e., a process instance) and a process model is a pairwise comparison between the executed tasks and the tasks allowed by the model. This pairwise comparison can generate three results: (i) perfect alignment step, i.e., the log is replayed correctly on the process model; (ii) missing events, i.e., the process model forces a task that is not recorded in the event log to be executed; (iii) wrong events, i.e., the process model does not allow a task recorded in the log to be executed.

In this step, we aim to answer the following questions: Is and how much the actual process of customer order fulfillment as recorded in the event log conforms to the standard process model designed by the company? In case there are discrepancies in the customer order fulfillment process where are they located? To answer these questions, first, we converted the standard model of customer order fulfillment process into a Petri net model. Then we computed the fitness metric by running "Conformance Checker" plugin in ProM. After that, we run the plugin "Conformance Checking of DPN (Xlog)" to locate discrepancies based on trace alignment. The value of fitness obtained is 0.945 which indicates that 94.5% of the recorded events are correctly replayed by the event log. This means 5.5% of the recorded events deviate from the standard model. The capture screen of trace alignment generated using the "Conformance Checking of DPN (Xlog)" plugin is depicted in Figure 2.5. Perfect alignments are illustrated in green color, missing events in purple color, and wrong events in yellow color.

The result of the alignment is summarized in Table 2.4. As can be seen, from 1653 replayed cases, there are 1460 cases where the event log and the process model are synchronized, i.e., 88.32% of the total executed cases followed exactly the standard process model designed by the company. This value is high; however, the existence of some missing and wrong events illustrates the presence of some deviations. According to the result obtained in Table 2.4, the activity N0160 has been allowed by the process model to be executed 154 times when it was not recorded in the event log. This can be seen as a skipped activity and indicates that 9.32% of the received customer orders skipped the post-processing painting and plating. Moreover, Table 2.4 shows that the activity N0180 has been executed 39 times while it was not allowed to be executed in the process model. This indicates that in 2.36% of total cases, the task operating on product delivery was performed several times in the same case. This might be explained by the fact that the task related to product delivery faced some issues and the task sent back to solve the problem occurred. This can be classified as a normal behavior. However, the fact that the post-processing painting and plating of the products has been skipped 154 times generates the following question. Are there special products that do not require the post-processing of painting and plating? If the answer to this question is ‘NO’, then an investigation about this observed negative behavior is required. A product, which necessitates painting and plating operations after quality inspection, skips this post-processing, might engender issues related to customer satisfaction. Nevertheless, if the answer to the above question is ‘YES’, then the standard process model requires an improvement. The process model discovered in the previous section can be guidance for the improvement of the standard process model. Excepting the question related to the post-processing task, we can say that in overall the customer order fulfillment process is well managed by the company.

Table 2.4. Result of an alignment between the executed activities and the activities allowed by the standard model

Alignment Result	**Number of traces**	**Model**	**Event log**	**Deviating tasks**	**% From total cases**
Perfect alignment	1460	√	√	-	88.32%
Missing Events	154	√	-	N0160	9.32%
Wong Events	39	-	√	N0180	2.36%

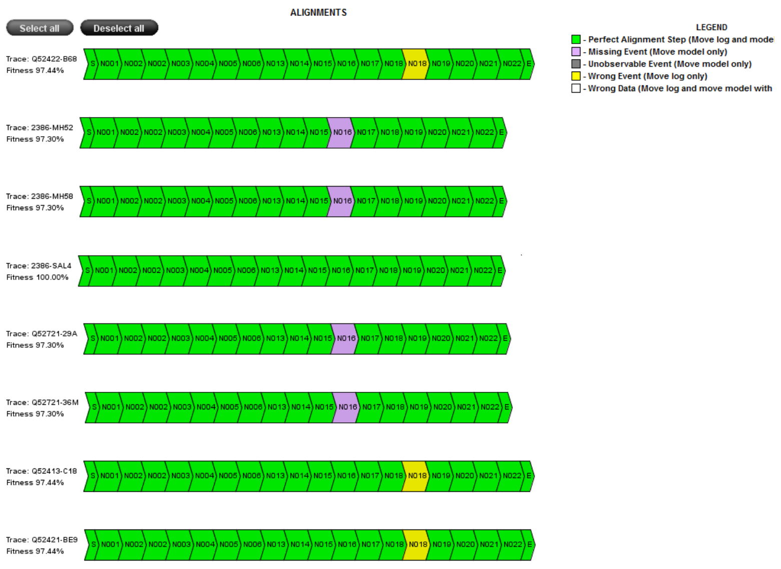

ALIGNMENTS
Select all
Deselect all
LEGEND
🟩 - Perfect Alignment Step (Move log and mode
🟪 - Missing Event (Move model only)
⬛ - Unobservable Event (Move model only)
🟨 - Wrong Event (Move log only)
⬜ - Wrong Data (Move log and move model with
Trace: Q52422-B68 Fitness 97.44%
S N001 N002 N002 N003 N004 N005 N006 N013 N014 N015 N016 N017 N018 N018 N019 N020 N021 N022 E
Trace: 2386-MH52 Fitness 97.30%
S N001 N002 N002 N003 N004 N005 N006 N013 N014 N015 N016 N017 N018 N019 N020 N021 N022 E
Trace: 2386-MH58 Fitness 97.30%
S N001 N002 N002 N003 N004 N005 N006 N013 N014 N015 N016 N017 N018 N019 N020 N021 N022 E
Trace: 2386-SAL4 Fitness 100.00%
S N001 N002 N002 N003 N004 N005 N006 N013 N014 N015 N016 N017 N018 N019 N020 N021 N022 E
Trace: Q52721-29A Fitness 97.30%
S N001 N002 N002 N003 N004 N005 N006 N013 N014 N015 N016 N017 N018 N019 N020 N021 N022 E
Trace: Q52721-36M Fitness 97.30%
S N001 N002 N002 N003 N004 N005 N006 N013 N014 N015 N016 N017 N018 N019 N020 N021 N022 E
Trace: Q52413-C18 Fitness 97.44%
S N001 N002 N002 N003 N004 N005 N006 N013 N014 N015 N016 N017 N018 N018 N019 N020 N021 N022 E
Trace: Q52421-BE9 Fitness 97.44%
S N001 N002 N002 N003 N004 N005 N006 N013 N014 N015 N016 N017 N018 N018 N019 N020 N021 N022 E

Figure 2.5. Captured screen of trace alignments illustrating perfect alignment, missing events and wrong events

2.2.7. Process performance analysis

In this section, the process model discovered in Section 2.5 is filtered by removing non-frequent paths forming a process model representing 99% of cases. This model is then extended with the timestamps of both activities and between activities, i.e., the time between the ending time of the preceding activity and the starting time of the following activity. With the time attribute of events, bottlenecks which negatively affect the performance of the whole process can be identified. For instance, the time attribute of activities enables to identify, in which activities, the process is spending too much time, while the timestamp between activities can allow detecting queuing delays in the process. The extended process model is shown in Figure 2.6. The numbers placed bellow activities depict the average time in which these activities are performed and the numbers associated with the transitions (i.e., arcs) represents the average time between activities, i.e., queuing delays. The thickness of transitions shows the average time between activities in comparison with the other arcs. The more a transition is thick; the longer is the time between activities.

As can be seen in Figure 2.6, there are three significant activities which are taking too much time in the whole process compared to others, i.e., N0130, N0160 and N0170. In other words, the production, the post-processing painting and plating and the packing sub-processes are taking in average 46.9, 37.5 and 47.2 days respectively. Regarding the time between activities, most of the transitions are automatic, i.e., they are instant or taking milliseconds. Nevertheless, there is a very thick transition between N0060 and N0130, which indicates that after requesting the purchase of raw materials required for production, it takes 47.6 days in average before the production of a customer order can start. In order to determine the reason why it takes this much time to start the production process, we have investigated the Raw Material Purchasing Process.

Raw material purchasing process is handled through 7 activities. This process starts by requesting a material purchase. The requested purchase is then approved and the approval of the purchase request is verified by the purchase department and then by the business department. As soon as it is verified, the purchase is finalized and the order of the purchase is performed. The last activity is receiving and verifying the ordered purchase. The attributes of these activities are presented in Table 2.5.

Similar to the customer order fulfillment process, we have extracted the data related to raw material purchase process from the SPPM system of the company. This data have been sorted, pre-processed, converted into an event log of XES format and filtered such that incomplete events are removed. The final obtained event log contains 9 activities including one start and one end, 663 cases and 9829 complete events recorded within one year and seven months. We have mined the model of raw material purchasing process using the fussy miner due to its strong ability in handling less structured process models. Figure 2.7 shows the derived model that represents the actual behavior of raw material purchasing process. As can be seen in Figure 2.7(a), we have obtained a spaghetti model containing frequent and infrequent paths that are difficult to understand. Therefore, this model has been filtered such that all activities and only frequent paths are maintained as shown in Figure 2.7(b). This process model is then extended in Figure 2.7(c) with the timestamps of activities and between activities for the purpose of investigating the performance characteristics of raw material purchasing process.

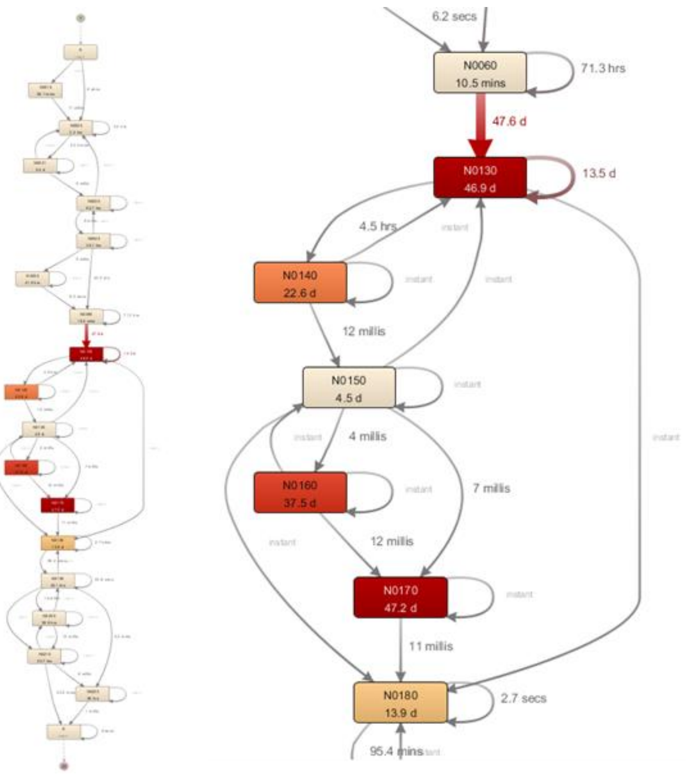


Figure 2.6. The mined process model extended with timestamps

As can be seen, most of the activities are taking few seconds or few minutes, except the activities N0100, N0090, N0080 and N0110 which are taking 38.1, 34.6, 22.8 and 15.1 hours respectively. In other words, purchase approval verification by purchase and business departments, purchase finalization and purchase ordering are taking a long time compared to other activities. Concerning the time between activities, we have identified several transitions that are taking too much time and which are all of them backward works. The performance and frequency of these transitions are presented in Table 2.6. The transitions N0070→S, N0080→S and N0090→S illustrate the situation where the purchases are not approved in different stages of the process, so sent back for verification and then to restart the purchase process of the material. The cause of these backward executions could be due to the lack of information, i.e., specifications, exact quantity, etc. The transitions E→N0090, E→N0100 and E→N0110 illustrate the situation where the ordered purchase is received, however, instead of forwarding it to the production process, the received material is sent back for purchase verification, purchase finalization and purchase reordering. In case it is not approved, the purchase is sent back to restart the purchase process. The cause of these backwards could be due to non-conformance of the received material with the desired material in terms of quality, specifications, etc., or due to mistakes committed during the purchasing process. Moreover, the arcs of these backward executions are thick which indicates that it takes too much time before the reworks (restart purchasing process, reapprove purchase, reorder purchase, etc.) can be performed.

Table 2.5. The attributes of activities of raw material purchasing process

Activity	**Attribute**
Start	S
Material Purchase Request	N0060
Purchase Approval	N0070
Purchase Approval Verification (Purchase Dep.)	N0080
Purchase Approval Verification (Business Dep.)	N0090
Purchase Finalization	N0100
Purchase Order	N0110
Materials Reception and Verification	N0120
End	E

Table 2.6. Backward transitions performance

Backward transitions	**Waiting time**	**Frequency**
N0070→S	49.6 hours	117 (17.64%)
N0080→S	46.7 hours	133 (20.06%)
N0090→S	5.6 days	53 (8%)
E→N0090	3.5 days	97 (14.63%)
E→N0100	3.2 days	100 (15.8%)
E→N0110	43.2 hours	27 (4%)

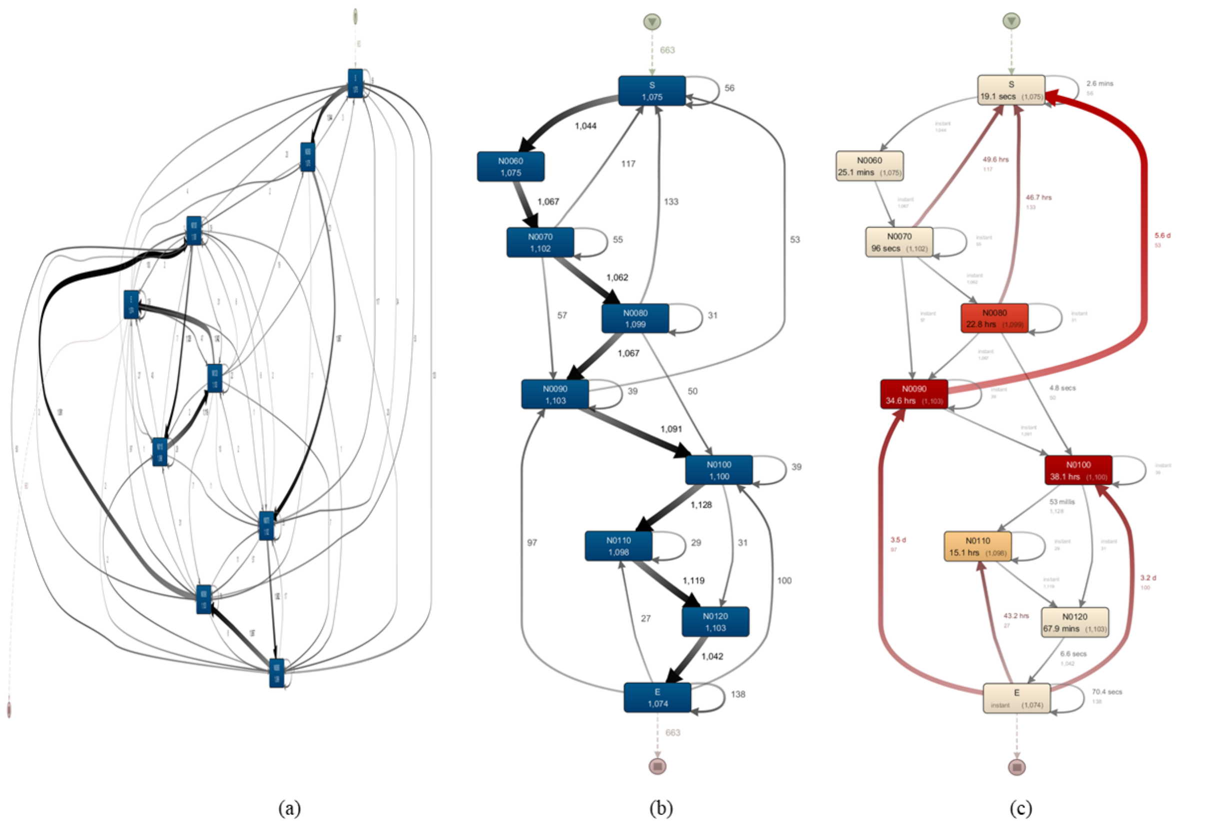


Figure 2. 7. Discovered model of materials purchasing process, i.e., a (100% activities, 100% paths detail), b (100% activities, 25% paths detail – only most important flows are shown) and c (100% activities, 25% paths detail, extension with timestamps)

Based on the identified activities that are taking too much time, the reworks, and the time spent before the reworks are performed and taking into consideration their frequencies, we can say that too many delays are occurring which negatively affect the performance of raw material purchasing process. Since the production process strongly depends on the availability of raw materials, any delay occurring in the material purchasing process engenders the delay of the production process. Therefore, in order to minimize the waiting time identified between requesting raw material purchases and the production process in the customer order fulfillment process, efficient actions need to be undertaken to reduce the backward executions and minimize the time of the identified activities that are taking too much time in the raw material purchasing process. In case a backward execution is necessary for this process, the waiting time to restart the process also need to be reduced.

Reducing the waiting time between requesting raw material purchases and the production process will not only accelerate the launching of production but will also contribute to reducing the lead time of the whole process of the customer order fulfilment process. Furthermore, to minimize more the lead time of the whole process of customer order fulfilment, production lead time, post-processing lead time, and packing lead time need to be reduced. It is because in these three sub-processes the customer order fulfilment process is spending too much time. As lead time reduction is considered one of the main focuses of lean manufacturing, implementing the principle of this approach can largely help in improving the performance of the customer order fulfilment process.

Conclusion

In this chapter, we presented a case study of applying process mining on a real-life customer order fulfillment process of a ship and naval parts manufacturing company, which is characterized by a very long lead time. The process under this case study refers to the entire process from the point a customer make an order, passing by order handling, production, quality inspection until the delivery of the final product to the customer. The data of this process have been extracted from the Shipbuilding Processing Plan Management System of the organization, preprocessed and converted into an event log supported by process mining tools. Then, we performed the discovery of the real as-is process model, followed by the conformance verification analysis. After that, we measured the actual performance of the customer order fulfillment process.

Process performance analysis has identified the existence of significant activities (sub-processes) taking too much time in the process as well as the existence of one major queuing delay between requesting material purchase and the production process. After requesting the purchase of raw materials required for production, it takes too much time (47.6 days in average) before the production of a customer order can start. In order to determine the reason why it takes this much time to start the production process, we have investigated the Raw Material Purchasing Process. By doing so, we found that too many delays are occurring which negatively affect the performance of raw material purchasing process. Since the production process strongly depends on the availability of raw materials, any delay occurring in the material purchasing process engenders the delay of the production process. Therefore, the first step to improve the performance of the customer order fulfillment process and reduce its lead time is to reduce the waiting time identified between requesting raw material purchases and the production process. This can be done by investigating the root causes of the delays occurred in the material purchasing process. The delays that need to be investigated include the activities where the material purchasing process is spending too much time, several reworks, and the waiting time before the reworks start. The second step to reduce the lead time of the customer order fulfillment process is to reduce the lead time of the major sub-processes which are affecting negatively the performance of the customer order fulfillment process. The major sub-processes where the whole process is spending too much time are the production, post-processing, and packing. As the lead time reduction is considered one of the main focuses of lean manufacturing, implementing the principle of this approach can largely help in improving the performance of the customer order fulfillment process. Now, we can clearly observe the importance of process mining in improving industrial business process.

Chapter 3 PRELIMINARIES

In this section, we define an event log, the starting point of process mining. We also define Petri net graphs and workflow net graphs which we will use as a process model representation. Moreover, we present preliminaries about the standard constructs and complex constructs of a workflow net.

3.1. Event logs

An event log stores the execution history of a business process. An example of event log is illustrated in Table 3.1. The events in the table are grouped by case and sorted chronologically. The sequence of events that is recorded for a process instance is called a trace. In this example, the trace for case 1 of Table 3.1 is <*request purchase*, *approve purchase*, *verify approval*, *finalize purchase*, *purchase order, receive purchase & verify*>. For a convenient use of events, we use a single letter for each activity instead of its full name. For instance, using single letters, the trace for case 1 will be $< a, b, c, d, e, f>$. An event log is a set of traces. Accordingly, the event log of Table 3.1.1 can be represented as $\{ < a, b, c, d, e, f>^{100}, < a, b, c, c, d, e, f >^{75}, \ldots\}$. For the rest of the thesis, we will use the single letters as an abbreviation for activities name.

Table 3.1. Example of an event log

Case id	Event id	Activity	Timestamp	….
Q521-QZR	N0060	Request Purchase	5/15/2014 16:35	….
	N0070	Approve Purchase	5/15/2014 16:40	….
	N0080	Verify Approval	5/16/2014 16:40	….
	N0090	Finalize Purchase	5/16/2014 17:42	….
	N0100	Purchase Order	5/16/2014 17:30	….
	N0110	Receive Purchase & verify	6/13/2014 10:55	….
Q523-B85	N0060	Request Purchase	5/15/2014 16:36	….
	N0070	Approve Purchase	5/15/2014 16:41	….
	N0080	Verify Approval	5/15/2014 16:41	….
	N0080	Verify Approval	5/15/2014 16:42	….
	N0090	Finalize Purchase	5/15/2014 16:55	….
	N0100	Order Purchase Order	5/15/2014 18:00	….
	N0110	Receive Purchase & verify	6/15/2014 16:44	….

.	.	.	.	.
.	.	.	.	.
				.

Definition 3.1 (*Trace, Event Log*)

Let $T \subseteq \pounds$ be the set of all process activities. An event e is the occurrence of an activity: $e \in T$. A trace $\sigma \in T^*$ is a sequence of activities. An event log L is a finite non-empty set of traces: $L \subseteq T^*$.

3.2. Process models

Process models represent the behaviour of a set of activities that belongs to a process or a workflow. Business process models are very primordial as it helps in having a deeper understanding of how your processes work and the way the business functions, so that the processes can be improved. Process models can be represented by several notations such as BPMN, transition systems, Petri nets, etc. In this thesis, we will use only Petri net representation to represent a process model. Petri net can be easily transformed to other notations.

3.2.1 Petri net

A Petri net is a directed bipartite graph that consists of places and transitions interconnected by directed arcs. Graphically, places are denoted by circles, transitions are represented by rectangles and arcs are represented by arrows with directly link places with transitions. Places represent possible states of the system; Transitions are events or actions which cause the change of state; and every arc simply connects a place with a transition or a transition with a place. A change of a state is denoted by a movement of *token(s)* (black dots) from place(s) to place(s); and is caused by the *firing* of a transition. The firing represents an occurrence of the event or an action taken. The firing is subject to the input conditions, denoted by token availability. The formal definition is presented in Definition 3.2. Figure 3.1 shows a petri net model deduced from the event log of Table 3.1.

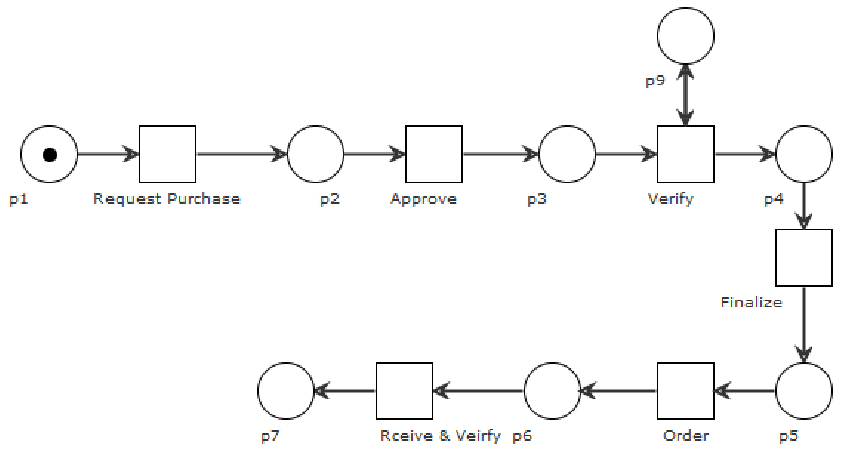


Figure 3.1. The corresponding petri net model of the event log of table 3.1(A sound workflow net)

Definition 3.2 (*Petri net graph*) (Van der Aalst, 1998).

A Petri net graph is a 3-tuple (P, T, A) in which P is a finite set of places; T is a finite set of transitions, such that $P \cap T = \emptyset$*; and* $A \subseteq (P \times T) \cup (T \times P)$ *is a set of directed arcs, called the flow relation. A place* $p \in P$ *is an input place of a transition* $t \in T$ *if and only if (iff) there exists an arc* $a \in A$ *such that* $\cdot t = \{p \in P / a(p, t) > 0\}$*. A place* $p \in P$ *is an output place of a transition* $t \in T$ *if there exists an arc* $a \in A$ *such that* $t \cdot = \{p \in P / a(t, p) > 0\}$.

A marking of a Petri net (graph). A marking of a Petri net is a multiset of its places, i.e., a mapping $M: S \to \mathcal{N}$, which means that the marking assigns a number of tokens to each place. A *marked Petri net* is a pair *(N, M)*, where *N = (P, T, A)* is a Petri net, and *M* is a bag over *P* denoting the marking of the net.

Definition 3.3 (*Firing rule*) (Van der Aalst, 1998).

Let (N = (P, T, A), M) be a marked Petri net. Transition $t \in T$ *is enabled (represented as* $(N, M)[t >)$ *iff* $\bullet t \leq M$*. The firing rule* $-[-> \subseteq N \times T \times N$ *is the smallest relation satisfying any (N = (P, T, A), M)* $\in \mathcal{N}$ *and any* $t \in T$*,* $(N, M)[t > \Rightarrow (N, M)[t > (N, M - \bullet t + t \bullet)$.

In the marking shown in Figure 3.1 (i.e., one token in the source place), transition *Request purchase* is enabled and firing this transition removes the token from the input place and puts a token in the output place. In the resulting marking, transition *Approve purchase* is enabled. If transition *Approve purchase* is executed, firing will removes the token from the input place of *Approve purchase* and puts a token in the output place of *Approve purchase* and so on.

Definition 3.4. (*Reachable markings*)

Let (N, s0) be a marked P/T-net in N. A marking s is reachable from the initial marking s0 iff there exists a sequence of enabled transitions whose firing leads from s0 to s. The set of reachable markings of (N, s0) is denoted [N, s0>.

The marked P/T-net shown in Figure 3.1 has 8 reachable markings. Sometimes it is convenient to know the sequence of transitions that are fired in order to reach some given marking.

Definition 3.5. (*Firing sequence*)

Let (N, s0) with N = (P, T, F) be a marked P/T net. A sequence $\sigma \in T^*$ is called a firing sequence of (N, s0) iff, for some natural number n $\in$ IN, there exist markings s1,...,sn and transitions t1,...,tn $\in$ T such that σ = t1 ...tn and, for all i with $0 \le i$.

Definition 3.6. *(Connectedness*)

A net N = (P, T, F) is weakly connected, or simply connected, iff, for *every two nodes x and y in P $\cup$ T, $x(F \cup F^{-1})^*y$, where R^{-1} is the inverse and R* the reflexive and transitive closure of a relation R. Net N is strongly connected iff, for every two nodes x and y, xF*y.*

We assume that all nets are weakly connected and have at least two nodes. The P/T-net shown in Figure 3.1 is connected but not strongly connected because there is no directed path from the sink place to the source place, or from *Receive & verify* to *Request purchase*, etc. (Van der Aalst, 2003).

Definition 3.7. (*Boundedness, safeness*)

A marked net (N = (P, T, F), s) is bounded iff the set of reachable markings [N, s> is finite. It is safe iff, for any s' $\in$ [N, s> and any p $\in$ P, s' (p) $\le$ 1. Note that safeness implies *boundedness.*

The marked P/T-net shown in Figure 3.1 is safe (and therefore also bounded) because none of the 8 reachable states puts more than one token in a place. (Van der Aalst, 2003).

Definition 3.8. (*Dead transitions, liveness*)

Let (N = (P, T, F), s) be a marked P/T-net. A transition t ∈ T is dead in (N, s) iff *there is no reachable marking s' ∈ [N, s> such that (N, s')[t . (N, s) is live iff, for every reachable marking s' ∈ [N, s> and t ∈ T, there is a reachable marking s" ∈ [N, s' such that (N, s")[t* >. Note that liveness implies the absence of dead transitions.

None of the transitions in the marked P/T-net shown in Figure 3.1 is dead. However, the marked P/T-net is not live since it is not possible to enable each transition continuously.

3.2.2 Workflow net

A workflow net is a special case of a Petri net that is used to model the workflow of process activities. Workflow net transitions represent activities or tasks, while places represent the pre/post conditions. A workflow net is a Petri net with a single source place and a single sink place. In addition, all nodes are on a path from start to end, except the source and sink. The model shown in Figure 3.1 is a workflow net. The formal definition of a workflow net is given as follows.

Definition 3.9.(*Workflow net graph*) (Van der Aalst, 1998).

(P, T, A) is a workflow net graph iff (P, T, A) is a Petri net graph, and there exists a single source place start $p \in P$ *such that* $\cdot p = \emptyset$*, and a single sink place end* $p \in P$ *such that* $p \cdot = \emptyset$*, and every place and every transition are on a path from start to end.*

Sound workflow net:

A workflow net is sound iff a process with a start marking of k tokens in its source place can reach the termination state marking with k tokens in its sink place (defined as k-sound workflow net). At the moment of termination, when there is a token in the sink place, all other places are empty. Additionally, in a sound workflow net, all transitions can fire (Van der Aalst, 1997). The workflow net shown in Figure 3.1 is a sound.

3.3. Standard constructs of a Petri net

A workflow net can be composed of the following standard constructs: sequence, AND-split, AND-join, XOR-split, and XOR-join. A workflow net with standard constructs is illustrated in Figure 3.2.

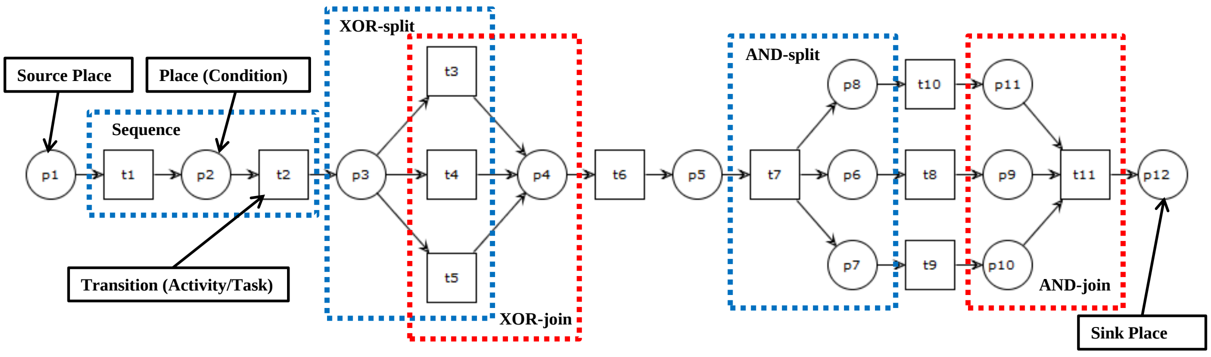


Figure 3.2. Example of a workflow net with standard constructs

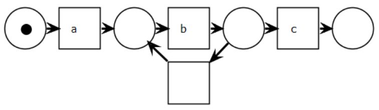
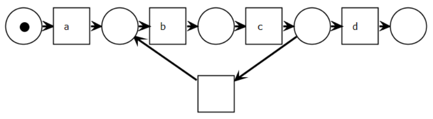
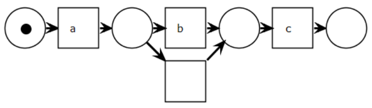
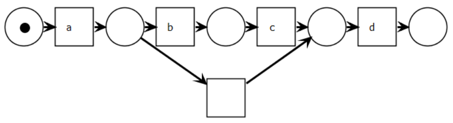
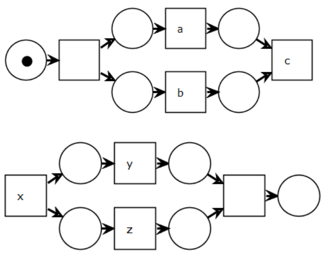
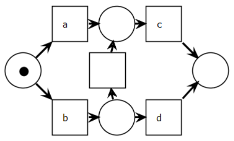
3.4. Complex constructs of a Petri net

The current process discovery algorithms are all capable of discovering sequences, choices, and parallelism. However, no existing technique can discover the following complex constructs together in a restricted time: short loops, invisible tasks, duplicate tasks and non-free choice constructs. In real-life processes, these complex constructs are common, so it is important to provide a technique that can successfully identify them all. We will define these complex constructs in this section before providing our discovery approach in the next section.

3.4.1. Invisible Tasks

An invisible task is a hidden task that has been executed but does not appear in the event log. For instance, the execution of a task that has been skipped to allow flexibility in the execution of a process is an invisible task that exists in the process model but cannot be recorded in the event log. Because invisible tasks are not present in the log, they are difficult to detect. Invisible tasks were first classified by (Wen *et al.*, 2007b) into four types according to the functional structure: SIDE, SKIP, REDO, and SWITCH. These four types are summarized in Table 3.2, as shown below. The abbreviation *IvT* will be used to refer to invisible tasks.

Table 3.2. Classification of invisible tasks

IvT types	**Definition**	**Event log**	**Process Model**
Short Redo (SR)	Repeat the execution of some tasks	{<a,b,c>, <a,b,b,c>}	
Long Redo (LR)		{<a,b,c,d>, <a,b,c,…b,c,d>}	
Short Skip (SS)	Skip the execution of some tasks	{<a,c>, <a,b,c>}	
Long Skip (LS)		{<a,d>, <a,b,c,d>}	
Side (SD)	Directly follow the source or sink place	{<a,b,c>, <b,a,c>} or {<x,y,z>, <x,z,y>}	
Switch (SW)	Switch the execution of some tasks	{<a,c>, <b,d>, <b,c>}	

3.4.2. Short Loops

A short loop can be either a loop of length one (L_{1p}) or a loop of length two (L_{2p}). In a loop of length one, there is one task that can be repeated several times, while a loop of length two involves two tasks that can be repeated many times. Fig. 3.3(a) depicts a process model with a loop of length one where task *B* can be repeated several times. Fig. 3.3(b) illustrates a process model with a loop of length two, where the two tasks *B* and *C* follow each other and can be repeated several times.

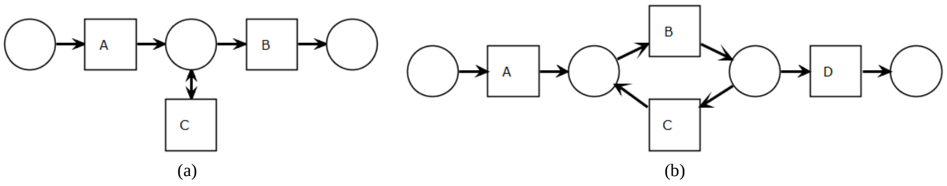


Figure 3.3. Example of sound process models with short loops: (a) a loop of length one and (b) a loop of length two.

3.4.3. Duplicate Tasks (DT)

Duplicate tasks are tasks sharing the same name but placed in two or more nodes in the workflow net. Fig. 3.4 depicts a process model with two duplicate tasks, *A* and *E*. Since it is difficult to differentiate between duplicate tasks sharing the same name in the event log, it is difficult to mine such a process model correctly based on its event log.

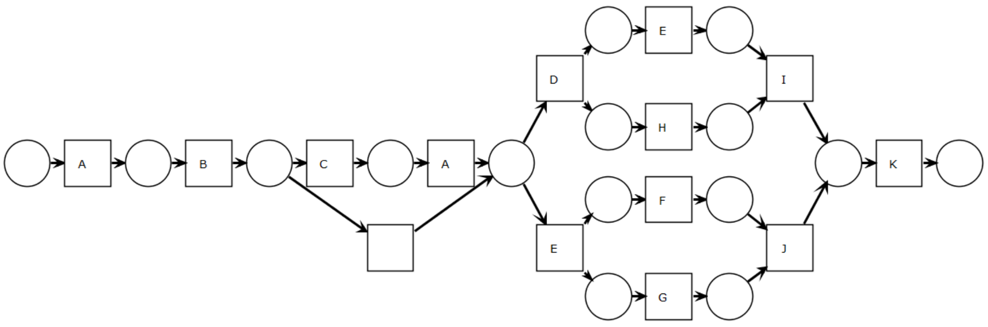


Figure 3.4. Example of a sound process model with duplicate tasks

3.4.4. Non-Free Choice Construct (NFC)

A non-free choice construct combines choice and synchronization (Wen *et al*, 2007a). In other words, the choice between two or more tasks depends on the tasks that have already been executed in a given process model. Fig. 3.5 gives an example of a workflow process model with a non-free-choice construct. In this model, if task *A* happens, task *D* will happen, and task *E* will not happen; whereas, if task *B* is executed, task *E* will be executed, but task *D* will not be executed. The existence of this non-free choice construct is due to the implicit dependencies between tasks *A* and *D* and between *B* and *E*. Therefore, it is necessary to detect all dependencies to correctly derive a process model with non-free choice constructs from an event log. There are two types of dependencies in workflow nets: explicit and implicit. An explicit dependency is a direct causal relationship between activities, while an implicit dependency is an indirect causal relationship between activities (Wen *et al*., 2006). The formal definitions of these two types of dependencies are given below.

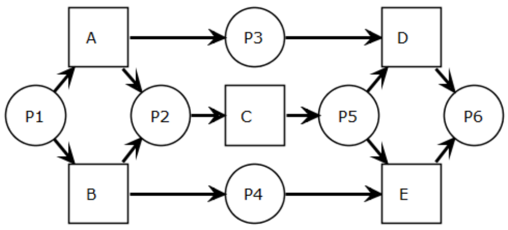


Figure 3.5. Sound workflow process model with a non-free choice construct

Definition 3.5 (*Explicit Dependency*) (Wen *et al*., 2006).

Let N = (P, T, A) be a sound workflow net with an input place i and an output place o. For any $a, b \in T$*, there is an explicit dependency between* a *and* b *iff:*

1. Connective: $a \bullet \cap \bullet b \neq \emptyset$*, and*

2. Successive: there is some reachable marking $s \in [N, [i] >$ *such that* $(N, s)[a >$ *and* $(N, s - \bullet a + a \bullet)[b >$.

Definition 3.6 (*Implicit Dependency*) (Wen *et al*., 2006).

Let N = (P, T, A) be a sound workflow net with an input place i and an output place o. For any $a, b \in T$*, there is an implicit dependency between* a *and* b *iff:*

1. Connective: $a \bullet \cap \bullet b \neq \emptyset$*,*

2. Disjunctive: there is no reachable marking $s \in [N, [i] >$ *such that* $(N, s)[a >$ *and* $(N, s - \bullet a + a \bullet)[b >$.

3. Reachable: there is some reachable marking $s \in [N, [i] >$ *such that* $(N, s)[a >$*, and there is some reachable marking* $s' \in [N, s - \bullet a + a \bullet>$ *such that* $(N, s')[b >$.

As shown in Figure 3.3.4, the place P_2 and its associated arcs represent explicit dependencies between *A* and *C* and between *B* and *C*, while P_3 and its associated arcs reflect an implicit dependency between *A* and *D*.

Conclusion

In this chapter, we introduced event logs and process models, and we discussed the Petri net and workflow net and also its standard and complex constructs because it is a language abstraction used by many process discovery techniques. We will use this model representation in this thesis, especially in Chapters 6 and7. In the next chapter, we will present our framework for process discovery algorithms.

Chapter 4 PROCESS DISCOVERY TECHNIQUES RECOMMENDATION FRAMEWORK

Process mining is new techniques whereby knowledge from event log stored in today's information systems are extracted to automatically construct business process models to have a full understanding of the real behaviour of processes, identify bottlenecks, and then improve them. Many process discovery algorithms have been proposed today. However, users and businesses still cannot choose or decide the appropriate mining algorithm for their business processes. Nevertheless, existing evaluation and recommendation frameworks have several important drawbacks. In this chapter, we propose a new framework for recommending the most suitable process discovery technique to a given process taking into consideration the limitations of existing frameworks.

4.1. Literature review

A number of papers have developed frameworks to evaluate or recommend process discovery techniques. Rozinat, J., et al. (2007) have written an important and a good starting article on building process mining algorithms benchmark framework. They introduced two methodologies to evaluate process discovery techniques. The first strategy is based on evaluation using metrics and the second one is a machine learning strategy. Wang, J., et al. (2012) extended this work by empirically evaluating process discovery techniques using artificial and real-life datasets, as well as different similarity measures. Despite the fact that this framework allow users and businesses select the process mining algorithm suitable to a given event log, empirical evaluation is time consuming. Only after performing experiments on all existing mining algorithms can be decided the best algorithm. Particularly, remarkable work do (Wang, J. et al., 2013) present in which the problem of spending long time in performing experiments to select the appropriate mining algorithm was considered. Their proposed framework is based on learning step and recommendation step. In the first phase a regression model is built based on features extracted from high quality selected reference models and on the similarities between the reference models and the mined models. Using this regression model as well as features extracted from the other models, similarities between reference models and mined models are predicted without performing experiments. While the authors' results obtained by applying this framework are very accurate and attractive, there is a weakness in this concept. The presented methodology is strongly based on reference models. However, reference models usually are not available in practical world. Ribeiro, J. et al. (2014) developed a framework and a tool for recommending control flow algorithms. Based on features extracted from event logs and prediction models built from experiments, top-K control-flow miners are recommended. Although this work takes event logs as input rather than reference models considering the limitation in (Wang, J. et al., 2013), the study can be criticized on the fact that in the framework, performing experiments to build prediction models are required before recommending top-k mining algorithm. Thus, again time consuming. The proposed systems would have been more interesting if the authors had based their framework on event logs and had reduced the experiments required to decide the mining algorithm suitable for a given event log. Pérez-Alfonso et al, (2015) proposed an approach to recommend a process discovery algorithm based only the classification of event logs, but they just conceptualised the idea. Jouck, T., et al. (2018) proposed a classification based framework to evaluate the quality of process discovery algorithms. The starting point of this framework is the generation of random samples of process models artificially from a specified population of processes. For each model, a training log with fitting traces and a test log with both fitting and non-fitting traces are generated. The quality of process discovery algorithms is based on the capability of the algorithm in correctly classifying a trace representing real process behaviour as fitting and a trace representing non-related behaviour to the process as non-fitting. Nevertheless, similar to other methodologies, it is an empirical framework. A user needs to run experiments on all existing discovery technique until he can decide the best algorithm based on quality performance.

4.2. Process Discovery Recommendation Framework

Existing evaluation and recommendation frameworks allow users choose the best algorithm to a given process by comparing the performance of existing discovery algorithms empirically. While in fact, recommending a process discovery technique for a given process log based on empirical assessment is time and resource consuming. From a business perspective, practically one cannot perform experiments on all algorithms each time to decide the most suitable algorithm at the end. In addition, some recommendation frameworks are based on reference models. There is a strong possibility that reference models are not available in the practical world.

The development of a recommendation framework for process discovery algorithms is strongly needed due to the abundance of process discovery algorithms and due to the fact that each algorithm has different characteristics and ability, and specific limitations. One of the big differences between existing discovery algorithms is the ability to discover the standards and complex constructs of a process model which are short loops, invisible tasks, non-free choice constructs, non-free choice involved in invisible tasks, and duplicate tasks. There is currently no algorithm that can handle all of these structures in a restricted time. For instance, the Inductive Miner algorithm (Leemans *et al.*, 2013) is robust for invisible tasks, but it cannot handle duplicate tasks and non-free choice constructs. Another example is the Region-based algorithm (Bergenthum et al., 2007), which is capable of mining some non-free choice constructs but cannot handle invisible and duplicate tasks. The other algorithms have similar problems. Each algorithm has an advantage in mining specific structures but at same time there is a restriction in mining other constructs.

In this chapter, we propose a new framework to recommend or select a mining algorithm suitable for a given event log. The idea of this framework consists of extracting the constructs of a process model from the corresponding event log without discovering any model. In other words, investigating the log whether it contains short loops, invisible tasks, non-free choice constructs, non-free choice involved in invisible tasks, and duplicate tasks. And using knowledge data base which contains the information about the capability of each algorithm in discovering the said structures, one can decide the candidate techniques suitable for the given event log. Knowledge data base can be constructed from the results obtained through existing comparison and evaluation frameworks that evaluate the ability of process discovery algorithms in mining the aforementioned constructs. In the proposed framework, no reference model and no empirical evaluation are needed to recommend a process discovery technique. In case the constructs extraction stage resulted in too many candidate algorithms, other metrics gathered in the knowledge data base from research papers (e.g. mining time, soundness, etc.) will be used to reduce the candidate algorithms. For instance, if after structures extraction from event logs, the results show that the log contains duplicate tasks, alpha algorithm, inductive miner and other algorithms will be eliminated due to their inability to discover duplicate tasks. Since Alpha* algorithm and genetic miner both can mine duplicate tasks, they are both candidates. However, according to the mining time metric in data base knowledge, genetic algorithm takes a long time compared to alpha* algorithm to discover a model from an event log characterized by the existence of duplicate tasks. In this case, the recommended discovery algorithm to mine such event logs is alpha* algorithm without performing any empirical evaluation. The overall framework is illustrated in Figure 4.1.

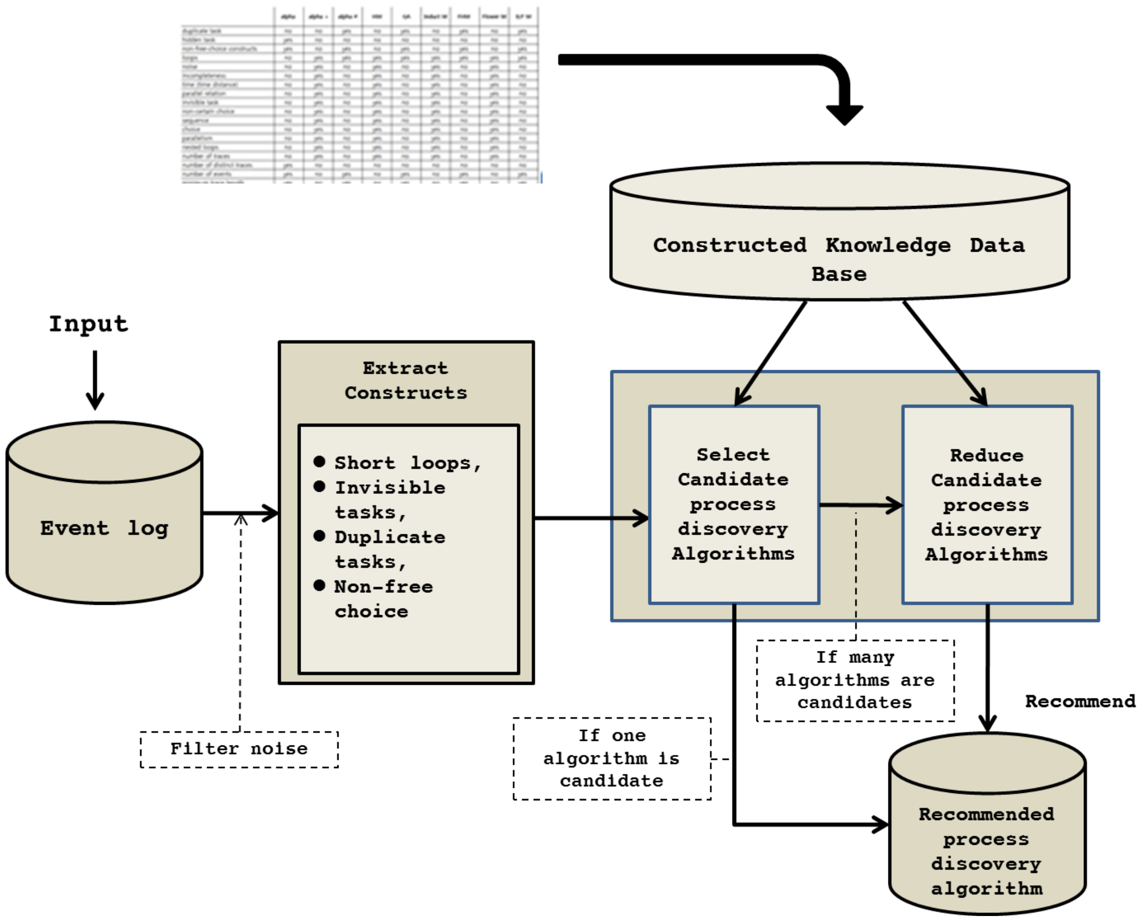

Constructed Knowledge Data Base
Input
Event log
Extract Constructs
● Short loops,
● Invisible tasks,
● Duplicate tasks,
● Non-free choice
Select Candidate process discovery Algorithms
Reduce Candidate process discovery Algorithms
If many algorithms are candidates
Recommend
Filter noise
If one algorithm is candidate
Recommended process discovery algorithm

Figure 4.1. Process discovery algorithms recommendation framework

4.2.1. Constructs

The most relevant constructs typically discovered by process discovery algorithms are sequence, exclusive choice, inclusive choice, parallelism, loops, invisible tasks, non-free choice, and duplicate tasks.

- ✓ **Sequence**: Certain process activities need to be sequentially executed.
- ✓ **Exclusive choice**: Certain process parts of the process are mutually exclusive. In several notations, this is known as XOR split/join.
- ✓ **Parallelism**: Certain branches are "parallel", indicating that the activities of a first part of the model are executed simultaneously with the activities of a second part of the model. In several notations, this is known as AND split/join.
- ✓ **Inclusive choice**: a choice needs to be made on which part(s) of process that follow

need to be performed, when reaching given points of the process. Inclusive choice is different from exclusive choice because multiple parts can be executed in parallel and different from the parallelism construct since not every part that follows the reached point needs to be executed. In several notations, this is known as OR split/join.

- ✓ **Loop**: The execution of certain parts of the process can be sequentially repeated multiple times.
- ✓ **Invisible tasks**: Certain transitions are incorporated into the model for a process-routing purpose. For instance, combined with exclusive choices, invisible tasks allow the execution of some parts of the process to be skipped.
- ✓ **Duplicate activities**: Certain activities share the same name but placed in two or more nodes in the process model.
- ✓ **Non-free choice**: when the choice of one or multiple branches is influenced by a choice occurred before.

We classify these constructs into standards constructs that can be discovered by all of existing process discovery algorithms and complex constructs that cannot be mined by all of the algorithms. We are not concerned with the standards constructs. Our main focus, in this chapter, is on the complex constructs which make the ability of each algorithm different in terms of mining these constructs.

4.2.2. Complex construct detection

In this section, we will present the equations used to detect short loops, invisible tasks, non-free choice and non-free choice involved in invisible tasks in a given event log. The equations for detecting the complex constructs are based on the extensions of alpha miner. Compared to other algorithms, the extensions of alpha miner (i.e., alpha^{++}, alpha$^{\$}$, alpha$^{\#}$) address most of the construct separately, i.e., short loops, all types of invisible tasks and non-free choice constructs.

(a) Basic ordering relations

In α algorithm, the originator and the driver of the $\alpha -$extensions algorithms, six basic relations were defined: $>_L$, Δ_L, $\Diamond_L$, $\rightarrow_L$, $||_L$, and $\neq_L$. Relation $>_L$ represents two activities that can successively be executed. If the same activity is executed two or multiple times successively, then the direct succession refers to a loop with length-one structure. Relation Δ_L expresses a loop with length-two structure (e.g., *aba*). This relation is used also to distinguish length-2-loop from parallel routing. Relation $\Diamond_L$ refers to two way loop-2-length, which means two activities have the Δ_L relations between each other (e.g. *aba, bab*). Relation $\rightarrow_L$ shows the direct causal relation between two activities. Relation $\|_L$ represents the activities that can be executed concurrently (e.g., *ab*, *ba*). Relation $\neq_L$ refers to two activities never following each other directly.

Definition 4.1(*Basic Ordering relations)*

Let A be a set of activities, L be an event log over A, a and b be two activities in A, the relation defined in α and α^+ algorithms as follows:

- $a >_L b \Leftrightarrow \exists \sigma = t_1 t_2 \ldots t_n \in L, i \in 1, \ldots, n-1: t_i = a \wedge t_{i+1} = b,$
- $a \, \Delta_L \, b \Leftrightarrow \exists \sigma = t_1 t_2 \ldots t_n \in L, i \in 1, \ldots, n-2: t_i = t_{i+2} = a \wedge t_{i+1} = b,$
- $a \Diamond_L b \Leftrightarrow (a \, \Delta_L \, b) \vee (b \, \Delta_L \, a),$
- $a \rightarrow_L b \Leftrightarrow \big((a >_L b) \wedge (b \not>_L a)\big) \vee (a \Diamond_L b),$
- $a \|_L b \Leftrightarrow (a >_L b) \wedge (b >_L a) \wedge (b \not\Diamond_L a),$ *and*
- $a \neq_L b \Leftrightarrow \big((a \not>_L b) \wedge (b \not>_L a)\big).$
- $a >_L a \Leftrightarrow \exists \sigma = t_1 t_2 \ldots t_n \in L, i \in 1, \ldots, n-1: t_i = a \wedge t_{i+1} = a,$

(b) Short loops detection

The relation that can detect loops of length two is defined in the basic ordering relations in Definition 4.1 as Δ_L. The relation that can detect loop of length one loop is defined using relation of direct succession as follows:

- $a >_L a \Leftrightarrow \exists \sigma = t_1 t_2 \ldots t_n \in L, i \in 1, \ldots, n-1: \; t_i = a \; \wedge \; t_{i+1} = a$

Several constructs and sound models cannot correctly be discovered with the existence of a loop of length-one. Therefore, most of the algorithms focus on the pre and post-processing steps of process mining as a solution to tackle length-one loops. Similarly, our framework starts by investigating the existence of a loop of length-one in the event log using the direct succession relation $>_L$. Once it is detected, the framework records the existence of loop of length-one, removes it from the event log and then investigates the rest of the constructs in the new event log defined in Definition 4.2.

Definition 4.2 (*loop-1-length free event log*)

Let A be a set of activities, L be an event log over A, a be an activity from A, and L1L be the set of loops of length-one, the new event log A' free of loop-1-length is defined as follows:

- $A_{log} = \{a \in A | \exists_{\sigma \in \mathrm{L}} [t \in \sigma]\}$,
- $L1L = \{ a \in A | \exists_{\sigma = \sigma = t_1 t_2 \ldots t_n \in L;\, i \in 2,3,..,n}\ a = t_i \wedge a = t_{i+1} \vee\ a = t_{i-1} \wedge a = t_i\}$,
- $A' = A_{log} \backslash L1L$.

(c) Invisible tasks detection

$\alpha^{\#}$ algorithm introduced the mendacious dependency $\rightsquigarrow_L$ to reflect invisible tasks of type SKIP, REDO, and SWITCH. The relation $\rightsquigarrow_L$ can be used to detect the existence of invisible tasks in the event log and to discover them in a process model. The mendacious dependency $\rightsquigarrow_L$ defined in $\alpha^{\#}$ algorithm is defined in Definition 4.3.

Definition 4.3 (*mendacious dependency associated with invisible tasks*)

Let A be a set of activities, L be an event log over A, and a, b be two activities from A. The mendacious dependency associated with invisible tasks is defined as follows:

- $a \rightsquigarrow_L b \Leftrightarrow (\mathrm{a} \rightarrow_L b) \wedge \exists\, x, y \in A{:}\, (\mathrm{a} \rightarrow_L x) \wedge (\mathrm{y} \rightarrow_L b) \wedge (y \not>_L x) \wedge (x \nparallel_L b) \wedge (a \nparallel_L y)$

This relation is capable of detecting invisible tasks of type *Short-Skip*, *Long-Skip*, *Short-Redo*, and *Long-Redo.* The basic idea of $\rightsquigarrow_L$ for detecting these types is illustrated in Figure 4.2. Task t in Figure 4.2 represents an invisible task. If the two tasks x and y are equal, t is of *Short-Skip* type. If x reaches y, t is of *Long-Skip* type. If the two tasks a and b are equivalent, t is of *Short-Redo* type. If b reaches a, t is of type *Long-redo* type. Otherwise, t is of type *Switch*. However, the relation detects these kinds of invisible tasks only when they are executed as a sequential workflow. The relation $\rightsquigarrow_L$ is incapable of detecting invisible tasks of these types if they are in parallel with other tasks. Therefore, $\alpha^\$$ algorithm improved the mendacious dependencies by introducing new definitions to detect invisible tasks of type *Short-Skip*, *Long-Skip*, *Short-Redo*, and *Long-Redo* when they are executed concurrently with other tasks.

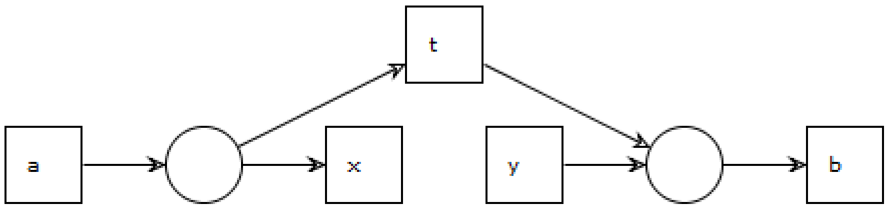


Figure 4.2. Illustration of the basic idea of $\rightsquigarrow_L$

$\alpha^\$$ algorithm introduced the *Between-set* to be used to improve the mendacious dependency. *Between-set* is defined in Definition 4.4, and refers to the tasks occurring between two tasks. If the two tasks are the endpoints of a concurrent construct, the *Between-Set* is the set of tasks in the parallel branches. For instance in Figure 4.3, $Between(L, a, b) = \{x, y, c\}$.

Definition 4.4 (*Between-Set*)

Let A be a set of activities, L be an event log over A, a and b be two activities from A, σ be a trace that belongs to L, the Between-Set of a,b, i.e. $Between(L, a, b)$*, is defined as follows:*

- $Between(\sigma, a, b) = \{\sigma_k | \exists_{1\le i<j\le m}(\sigma_i = a \wedge \sigma_j = b) \wedge i < k < j \wedge \nexists_{i<l<j}$ $(\sigma_l = a \vee \sigma_l = b))\}$,
- $\neg Between(\sigma, a, b) = \{\sigma_k | 1 \le k \le m\} \setminus Between(\sigma, a, b), and$
- $Between(L, a, b) = \cup_{\sigma\in L} Between(\sigma, a, b) \setminus \cup_{\sigma\in L} \neg Between(\sigma, a, b)$.

The mendacious dependency $\rightsquigarrow_L$ is improved as $\hookrightarrow_L$ and defined in Definition 4.5. the improved mendacious dependency $\hookrightarrow_L$ is capable of detecting the types of invisibles tasks when they are involved in a concurrent construct. The basic idea of $\hookrightarrow_L$ is illustrated in Figure

4.3. Task t in Figure 4.3 represents an invisible task in a parallel branch. Similar to Figure 4.2, if the two tasks x and y are equal, t is of *Short-Skip* type. If x reaches y, t is of *Long-Skip* type. If the two tasks a and b are equivalent, t is of *Short-Redo* type. If b reaches a, t is of type *Long-redo* type. Otherwise, t is of type *Switch*.

Definition 4.5 (*Mendacious dependencies associated with invisible tasks in parallel*)

Let A be a set of activities, L be an event log over A, a and b be two activities from A, σ be a trace that belongs to L, the mendacious dependency $a \hookrightarrow_L b$ associated with invisible tasks involved in a parallel construct is defined as follows:

- $a \geq_L b \Leftrightarrow \exists_{x,y \in A}\ Between\ (L,x,y) \subset Between\ (L,a,b) \wedge \forall m \in \big(Between(L,a,b) \setminus (Between(L,x,y) \cup \{x,y\})\big)\ \forall n \in Between\ (L,x,y)$
  $: (m \parallel_L n) \wedge \exists_{\sigma \in L} Between\ (\sigma,a,b) \subseteq (Between(L,a,b) \setminus \{x,y\}),$
- $a \rightrightarrows_L b \Leftrightarrow \big((a \geq_L b) \wedge (b \ngeq_L a)\big) \vee (a \between_L b)$, and
- $a \hookrightarrow_L b \Leftrightarrow \exists\, x,y \in A: (\mathrm{a} \rightarrow_L x) \wedge (\mathrm{y} \rightarrow_L b) \wedge (x \nparallel_L b) \wedge (y \nparallel_L a) \wedge (a \rightrightarrows_L b)$
  $\wedge (y \ngeq_L x).$

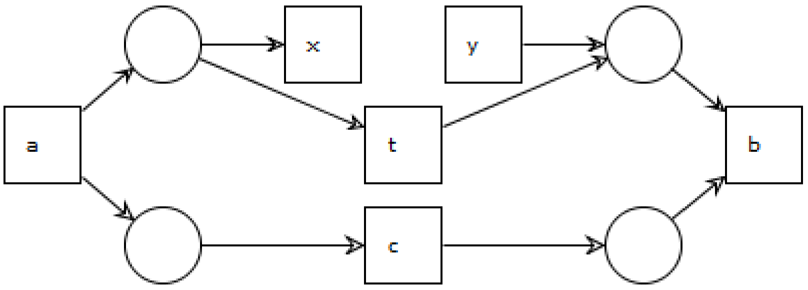


Figure 4.3. Illustration of the basic idea of improved mendacious dependency

The mendacious dependency $\rightsquigarrow_L$ can detect invisible tasks of types *Skip*, *Redo*, and *Switch* involved in a sequential constructs using one relation. The dependency $\rightsquigarrow_L$ can be seen as a combined relation that can detect invisible tasks of type *Short-Skip*, *Long-Skip*, *Short-Redo*, and *Long-Redo* involved in a sequential construct. Similarly, the improved one $\hookrightarrow_L$ can discover invisible tasks of types *Skip*, *Redo*, and *Switch* involved in a parallel construct using one relation. The dependency $\hookrightarrow_L$ can be seen as a combined relation that can detect invisible tasks of type *Short-Skip*, *Long-Skip*, *Short-Redo*, and *Long-Redo* involved in a parallel construct. In fact, there are process discovery algorithms which can detect a certain types of invisible tasks while cannot detect the other types. For instance, ETM algorithm (Buijs et al., 2012) is capable of discovering invisible tasks of types *Short-Skip* and *Long-Skip* involved in a sequential construct whereas it is incapable of discovering those of types *Short-Redo* and *long redo* in sequence, and *Short-Skip* and *Long-Skip* in parallel and switch. Similarly, the Heuristic Miner (Weijters *et al.*, 2006) is proven to be able to detect invisible tasks. However, it cannot detect those of types *Short-Redo* in a sequence construct and *Short-Skip* in a parallel construct. Hence, even if the Heuristic Miner is proven to discover invisible tasks, if the existence of invisible task of type *Short-Skip* in a parallel construct is detected in the log, the Heuristic Miner will not be recommended. The dependencies $\rightsquigarrow_L$ and $\hookrightarrow_L$ do not detect the types of invisible tasks separately. Therefore, we need to define a relation for each type of invisible tasks.

In our framework, we define a relation for each type of invisible tasks by splitting the two relations $\rightsquigarrow_L$ and $\hookrightarrow_L$ and based on the information that if the two tasks x and y are equal, t is of *Short-Skip* type. If x reaches y, t is of *Long-Skip* type. If the two tasks a and b are equivalent, t is of *Short-Redo* type. If b reaches a, t is of type *Long-redo* type. Otherwise, t is of type *Switch*. The relations detecting invisible tasks of types *Short-Skip*, *Long-Skip*, *Short-Redo*, and *Long-Redo* in a sequential construct is defined in Definition 4.7, Definition 4.8, Definition 4.9, and Definition 4.10 respectively. The relations detecting of those involved in a parallel construct are defined in Definition 4.11, Definition 4.12, Definition 4.13, and Definition 4.14 respectively.

Before that we define two relations will be used in the new definitions for detecting each type of invisible tasks separately. These two relations are $\gg_L$ and $\succ_L$. They were introduced in α^{++} algorithm. Relation $\gg_L$represents the case where one task can only be indirectly followed by another task, while relation $\succ_L$ refers to the situation when one task can be followed by another task either directly or indirectly. $a \gg_L b$ means that task b is reachable from task a indirectly and $a \succ_L b$ indicates that task b is reachable from task a directly or indirectly. Relations $\gg_L$ and $\succ_L$ are defined in Definition 4.6.

Definition 4.6 (*Reachable dependencies*)

Let A be a set of tasks, L be an event log over A, a, b two tasks from A, the dependency $a \gg_L b$ *reflects the indirect reachable dependency between the two tasks a and b and* $a \succ_L b$

reflects the indirect or direct reachable dependency between the two tasks a and b. These two relations are defined as follows:

- $a \gg_L b \Leftrightarrow \exists \sigma = t_1 t_2 \dots t_n \wedge i, j \in 1, \dots, n : i < j \wedge t_i = a \wedge t_j = b \wedge \forall k \in [i+1, \dots, j-1] : t_k \neq a \wedge t_k \neq b$
- $a \gtrdot_L b \Leftrightarrow (a \rightarrow_L b) \vee (a \gg_L b)$

Definition 4.7 (*Short skip in sequence*)

Let A be a set of tasks, L be an event log over A, a, b two tasks from A, the dependency $a \dashrightarrow_L^{sss} b$ *reflects the mendacious dependency associated with invisible tasks of type Short Skip in Sequence and is defined as follows:*

$$a \dashrightarrow_L^{sss} b \Leftrightarrow (a \rightarrow_L b) \wedge \exists x \in A: (a \rightarrow_L x) \wedge (x \rightarrow_L b) \wedge (x \nparallel_w b) \wedge (a \nparallel_w x) \wedge (x \not\gtrdot_L x).$$

Definition 4.8 (*Short redo in sequence*)

Let A be a set of tasks, L be an event log over A, x, y two tasks from A. the dependency $x \dashrightarrow_L^{srs} y$ *represents the mendacious dependency associated with invisible tasks of type Short Redo in Sequence and is defined as follows:*

$$\mathrm{y} \dashrightarrow_L^{srs} \mathrm{x} \Leftrightarrow (y \not\gtrdot_L x) \wedge \exists a \in T: (\mathrm{a} \rightarrow_L x) \wedge (\mathrm{y} \rightarrow_L a) \wedge (x \nparallel_L a) \wedge (a \nparallel_L y) \wedge (a >_L a).$$

Definition 4.9 (*Long skip in sequence*)

Let A be a set of tasks, L be an event log over A, a, b two activities from A. the dependency $a \dashrightarrow_L^{lss} b$ *reflects the mendacious dependency associated with invisible tasks of type long Skip in Sequence and is defined as follows:*

$$\mathrm{a} \dashrightarrow_L^{lss} \mathrm{b} \Leftrightarrow (\mathrm{a} \rightarrow_L b) \wedge \exists\, x, y \in A: (\mathrm{a} \rightarrow_L x) \wedge (\mathrm{y} \rightarrow_L b) \wedge (y \not\gtrdot_L x) \wedge (x \nparallel_L b) \wedge (a \nparallel_L y) \wedge (x \gtrdot_L y).$$

Definition 4.10 (*long redo in sequence*)

Let A be a set of tasks, L be an event log over A, x, y two activities from A. the dependency $x \dashrightarrow_L^{srs} y$ which represents the mendacious dependency associated with invisible tasks of type long Redo in Sequence is defined as follows:

$$y \dashrightarrow_L^{lrs} x \Leftrightarrow \exists\, a, b \in A\colon (a \rightarrow_L x) \land (y \rightarrow_L b) \land (x \not\#_L b) \land (y \not\#_L a) \land (b \gtrsim_L a) \land (a >_L b).$$

Definition 4.11 (*Short skip in parallel*)

Let A be a set of tasks, L be an event log over A, a, b two activities from A, the dependency $x \dashrightarrow_L^{ssp} y$ which represents the mendacious dependency associated with invisible tasks of type Short-Skip in Parallel is defined as follows:

- $a \geq_L^{sh} b \Leftrightarrow \exists x \in Between(L, a, b) \land \forall\, m \in (Between(L, a, b) \setminus \{x\})\colon (m \parallel_L x)$
  $\land\ \exists_{\sigma \in L} Between(\sigma, a, b) \subseteq (Between(L, a, b) \setminus \{x\})$,
- $a \rightrightarrows_L^{sh} b \Leftrightarrow \left((a \geq_L^{sh} b) \land (b \ngeq_L^{sh} a)\right) \lor (a \between_L b)$,
- $a \dashrightarrow_L^{ssp} b \Leftrightarrow \exists\, x, y \in A\colon (a \rightarrow_L x) \land (y \rightarrow_L b) \land (x \not\#_L b) \land (y \not\#_L a) \land (a \rightrightarrows_L^{sh} b)$.

Definition 4.12 (*Short redo in parallel*)

Let A be a set of tasks, L be an event log over A, x, y two activities from A, the dependency $x \dashrightarrow_L^{srp} y$ which represents the mendacious dependency associated with invisible tasks of type Short-Redo in Parallel is defined as follows:

$$y \dashrightarrow_L^{srp} x \Leftrightarrow \exists\, a \in A\colon (a \rightarrow_L x) \land (y \rightarrow_L a) \land (a \not\#_L x) \land (y \not\#_L a) \land \left(y \rightrightarrows_L^{sh} x\right) \land (a >_L a).$$

Definition 4.13 (*long skip in parallel*)

Let A be a set of tasks, L be an event log over A, a, b two activities from A, the dependency $x \dashrightarrow_L^{lsp} y$ which represents the mendacious dependency associated with invisible tasks of type Long-Skip in Parallel is defined as follows:

- $a \geq_L^{lg} b \Leftrightarrow \exists_{x,y \in A}\ Between\,(L,x,y) \subset Between\,(L,a,b) \wedge \forall m \in \big(Between(L,a,b) \setminus (Between(L,x,y) \cup \{x,y\})\big)\ \forall n \in Between\,(L,x,y) : (m \parallel_L n) \wedge \exists_{\sigma \in L} Between\,(\sigma,a,b) \subseteq (Between(L,a,b) \setminus \{x,y\}),$
- $a \rightrightarrows_L^{lg} b \Leftrightarrow \Big(\big(a \geq_L^{lg} b\big) \wedge \big(b \ngeq_L^{lg} a\big)\Big) \vee (a \lozenge_L b),$
- $a \dashrightarrow_L^{lsp} b \Leftrightarrow \exists\, x,y \in A\colon (\mathrm{a} \rightarrow_L x) \wedge (\mathrm{y} \rightarrow_L b) \wedge (x \nparallel_L b) \wedge (y \nparallel_L a) \wedge \big(a \rightrightarrows_L^{lg} b\big) \wedge (y \ngeq_L x) \wedge (x \gtrdot_L y).$

Definition 4.14 (*long redo in parallel*)

Let A be a set of tasks, L be an event log over A, a, b two activities from A, the dependency $x \dashrightarrow_L^{lrp} y$ *which represents the mendacious dependency associated with invisible tasks of type Long-Redo in Parallel is defined as follows:*

$$y \dashrightarrow_L^{lrp} x \Leftrightarrow \exists\, a,b \in A\colon (\mathrm{a} \rightarrow_L x) \wedge (\mathrm{y} \rightarrow_L b) \wedge (x \nparallel_L b) \wedge (y \nparallel_L a) \wedge \big(y \rightrightarrows_L^{lg} x\big) \wedge (b \ngeq_L a) \wedge (b \gtrdot_L a).$$

(d) Non-Free Choice constructs

Non-free choice constructs are detected by detecting implicit dependencies. α^{++} algorithm introduced three implicit dependencies to detect five types of non-free choice constructs illustrated in Figure 4.4. However, the proposed relations are based on the places component of a workflow net. Thus, they are based on both the discovered process model and the event log. In our framework, we are supposed to detect non-free choice constructs based only on the information in the event log. Therefore, we transformed the three implicit dependencies such that the conditions on places are converted to transitions. For more details about the implicit dependencies of defined in α^{++} algorithm, refer to (Wen *et al.*, 2007a). The transformed implicit ordering relations for detecting non-free choice constructs are defined in Definition 4.15.

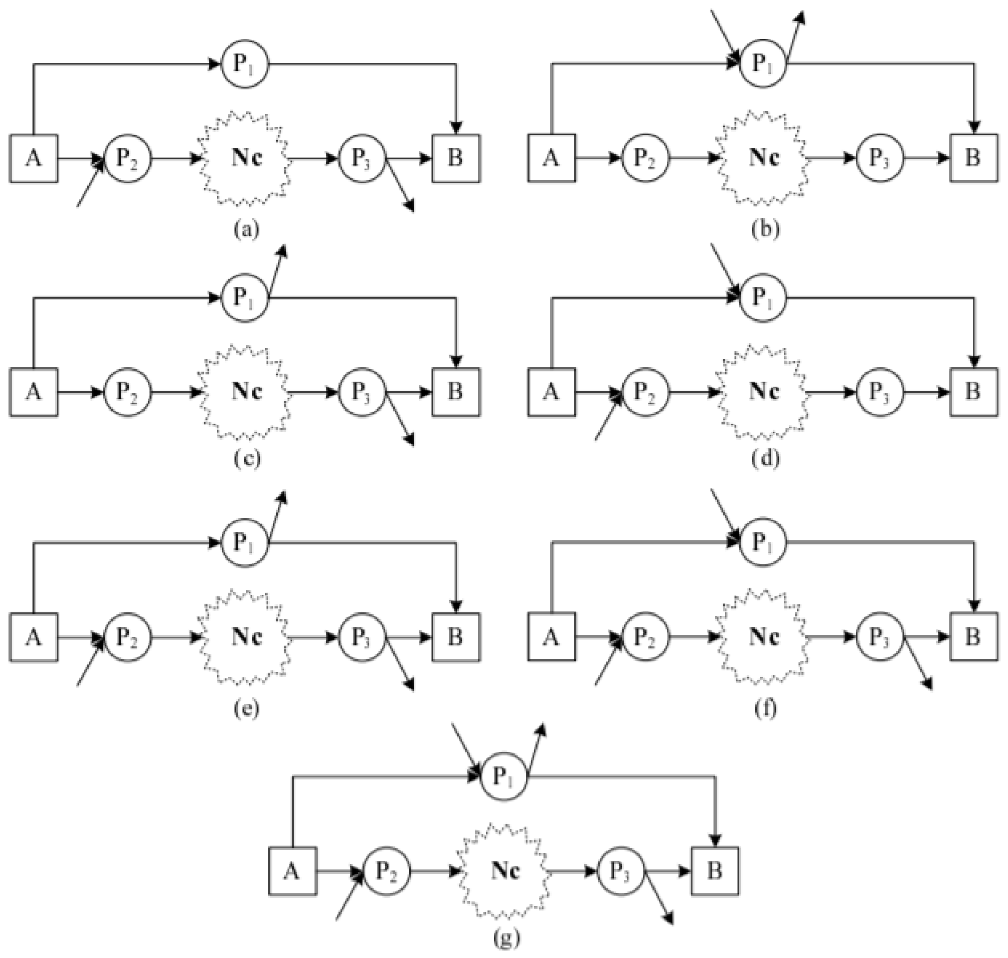


Figure 4.4. Sound sub-WF-nets with different location of implicit dependencies (Wen et al., 2006)

Before that we introduce two relations $\lhd_L$ and $\rhd_L$ defined in α^{++} algorithm and which will be used in the definition of implicit dependencies. Moreover, we introduce a new relation, the $\lhd\!\!\mid_L$, which we will use it too in defining the implicit dependencies. Relation $\lhd_L$ represents XOR-Split, $\rhd_L$ corresponds to XOR-Join and $\lhd\!\!\mid_L$ represent AND-split. Relations $\lhd_L$, $\rhd_L$ and $\lhd\!\!\mid_L$ are defined in Definition 4.15.

Definition 4.15 (XOR-Split, XOR-Join and AND-split)

Let A be a set of tasks, L be an event log over A, a, b two activities from A.

- $a \lhd_L b \Leftrightarrow (a \neq_L b) \wedge \exists c \in A: (c \rightarrow_L a) \wedge (c \rightarrow_L b),$
- $a \rhd_L b \Leftrightarrow (a \neq_L b) \wedge \exists c \in A: (a \rightarrow_L c) \wedge (b \rightarrow_L c),$

- $a \lhd\!\!|_L \Leftrightarrow \exists x, y \in A: (a \rightarrow_L x) \wedge (a \rightarrow_L y) \wedge (x \parallel_L y)$.

Definition 4.16 (*Implicit ordering relations*)

Let A be a set of tasks, L be an event log over A, a, b two activities from A, the implicit dependencies $\rightarrow_{L^1}$, $\rightarrow_{L^2}$, and $\rightarrow_{L^3}$ which detect non-free choice constructs are defined as follows:

- $a \rightarrow_{L^1} b \Leftrightarrow (a \not>_L b) \wedge \exists c \in A: (a >_L c) \wedge (c \lhd_L b) \wedge \nexists t \in A: (t >_L a) \wedge \big((t >_L a) \vee (t \parallel_L a)\big)$,
- $a \rightarrow_{L^{21}} b \Leftrightarrow (a \gg_L b) \wedge a \lhd\!\!|_L \wedge \exists b' \in A: (b \lhd_L b') \wedge \nexists t \in A: a >_L t \wedge (t \gtrsim_L b \text{ or } t \parallel_L b) \wedge \exists t' \in A: a >_L t' \wedge (t' {}^{\gtrsim_L} b \vee t' \parallel_L b)$,
- $a \rightarrow_{L^{22}} b \Leftrightarrow (a \gg_L b) \wedge b \,|\!\!\rhd_L \wedge \exists a'' \in A: (a \rhd_L a') \wedge \nexists t \in A: t >_L b \wedge (a \gtrsim_L t \text{ or } a \parallel_L t) \wedge \exists t' \in A: t' {}^{>_L} b \wedge (a' {}^{\gtrsim_L} t' \vee a' \parallel_L t')$,
- $a \rightarrow_{L^2} b \Leftrightarrow a \rightarrow_{L^{21}} b \vee a \rightarrow_{L^{22}} b$, and
- $a \rightarrow_{L^3} b \Leftrightarrow (a \lhd_L a') \wedge (b \rhd_L b') \wedge (a \gg_L b) \wedge (a \neg\!\gg_L b')$.

First of all, relation $\rightarrow_{L^1}$ detects the implicit dependencies illustrated in Figure 4.4(b) and (g) from an event log. Secondly, relation $\rightarrow_{L^2}$ detects the implicit dependencies shown in Figure 4.4(c) to (f). Finally, relation $\rightarrow_{L^3}$ detects the implicit dependencies similar to Figure 4.4(a).

(e) Invisible tasks involved in a non-free choice construct detection

There are cases where invisible tasks are involved in a non-free choice construct. An example is illustrated in Figure 4.5. Tasks *t1* and *t2* are invisible tasks designed to skip the execution of tasks *B* and *E*. *t2* together with *E*, p2, p3, and p4 form a non-free choice construct indicating that if the invisible tasks t1 is executed, t2 will be later executed and E will not be performed, and if B is executed, E will be executed later and not t2. To detect the involvement of invisible tasks in a non-free choice construct, the reachable dependencies between the two invisible tasks *t1* and *t2* need to be detected in this example. For this, $\alpha^{\$}$ algorithm introduced the notion of conditional reachable dependency (CRD) which requires adding artificially a starting task (i.e., $\perp$) and an ending task (i.e., $\top$). The conditional reachable dependency is defined in Definition 4.17. The relation a $\gg_L b$ which indicates that

a is indirectly followed by b is used in this definition. Definition 4.17 introduced three types of CRDs: pre-CRD (i.e., $\succ_{L,pre=x}$), post-CRD (i.e., $\succ_{L,post=y}$), and both-CRD (i.e., $\succ_{L,pre=x,post=y}$). a $\succ_{L,pre=x,post=y}$ b indicates that there is a trace where a $\gg_L$ b holds and x is executed directly before a and y is performed directly after b.

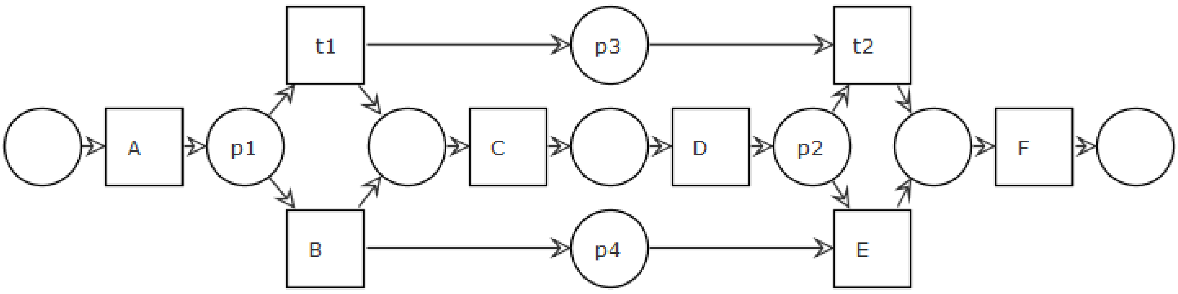


Figure 4.5. Example of a sound WF-net with invisibles tasks t1 and t2 involved in a non-free choice construct (L = {<A, B, C, D, E, F>, <A, C, D, F>})

Definition 4.17 (*Conditional reachable dependency*)

Let A be a set of activities, L be an event log over A, ⊥ and ⊤ are staring activity and ending activity artificially added to each trace in the event log such that for a trace σ with length n, $\sigma_0 = \perp$ *and* $\sigma_{n+1} = \top$*, a, b two activities from A and x, y two activities from* $A \cup \{\perp\} \cup \{\top\}$. Conditional reachable dependencies are defined as follows:

- a $\succ_{L,pre=x}$ b $\Leftrightarrow$ (a $\rightarrow_L b$) $\vee$ $(\exists_{\sigma \in L \wedge 1 \leq i \leq |\sigma|}\ \sigma_i = a\ \wedge\ \sigma_{i-1} = x\ \wedge (a \gg_\sigma b))$,
- a $\succ_{L,post=y}$ b $\Leftrightarrow$ (a $\rightarrow_L b$) $\vee$ $(\exists_{\sigma \in L \wedge 1 \leq j \leq |\sigma|}\ \sigma_j = b\ \wedge\ \sigma_{j+1} = y\ \wedge (a \gg_\sigma b))$,
- a $\succ_{L,pre=x,post=y}$ b $\Leftrightarrow$ (a $\rightarrow_L b$) $\vee$ $(\exists_{\sigma \in L \wedge 1 \leq i,j \leq |\sigma|}\ \sigma_i = a\ \wedge\ \sigma_j = b\ \wedge\ \sigma_{j+1} = y$ $\wedge\ \sigma_{i-1} = x\ \wedge (a \gg_\sigma b))$.

$\alpha^\$$ algorithm defined the reachable dependency related to invisible tasks based on conditional reachable dependency. For two invisible tasks x and y, $x \succ_L y$ holds if there exist four tasks a_1, a_2, b_1, and b_2 such that $a_1 \rightarrow_L x$, $x \rightarrow_L b_1$, $a_2 \rightarrow_L y$, $y \rightarrow_L b_2$, and $b_1 \succ_{L,pre=a_1,post=b_2} a_2$. $x \succ_L m$ holds if there exist two tasks a and b such that $a \rightarrow_L x$, $x \rightarrow_L b$, and $b \succ_{L,pre=a} m$. $m \succ_L x$ is similar to $x \succ_L m$. The reachable dependencies related to invisible tasks involved in a non-free choice construct are defined in Definition 4.18.

Definition 4.18 (*Reachable dependencies related to invisible tasks with non-free choice*)

Let A be a set of activities, L be an event log over A, me be an activity from A, x,y be two invisible tasks, the reachable dependencies related to invisible tasks in a non-free choice are defined as follows:

- $x \gtrdot_L m \Leftrightarrow \exists_{(a=\perp \vee a \in L) \wedge b \in L} (a \rightarrow_L x) \wedge (x \rightarrow_L b) \wedge b \gtrdot_{L,pre=a} m$
- $m \gtrdot_L x \Leftrightarrow \exists_{a \in L \wedge (b \in L \vee b = \top)} (a \rightarrow_L x) \wedge (x \rightarrow_L b) \wedge m \gtrdot_{L,pre=b} a$
- $x \gtrdot_L y \Leftrightarrow \exists_{(a_1=\perp \vee a_1 \in L) \wedge b_1 \in L \wedge a_2 \in L \wedge (b_2 \in L \vee b_2 = \top)} (a_1 \rightarrow_L x) \wedge (x \rightarrow_L b_1)$ $\wedge (a_2 \rightarrow_L y) \wedge (y \rightarrow_L b_2) \wedge b_1 \gtrdot_{L,pre=a_1,post=b_2} a_2$.

However, the reachable dependency defined in $\alpha^{\$}$ algorithm to detect the involvement of invisible tasks in a non-free choice construct actually does not differentiate between non-free choice and a free choice. For instance, $t1 \gtrdot_L t2$ of the event log of the WF-net shown in Figure 4.5 where there is a reachable dependency between $t1$and $t2$ holds, and $t1 \gtrdot_L t2$ of the event log of the WF-net shown in Figure 4.6 where there is no reachable dependency between $t1$and $t2$ also holds. In the log of Figure 4.6 $t1 \gtrdot_L t2$ holds and also $t1 \gtrdot_L E$ holds. To detect that there is reachable dependency between $t1$and $t2$ $t1 \gtrdot_L E$ should not hold. Based on this, we improved the reachable dependency related to invisible tasks in Definition 4.19.

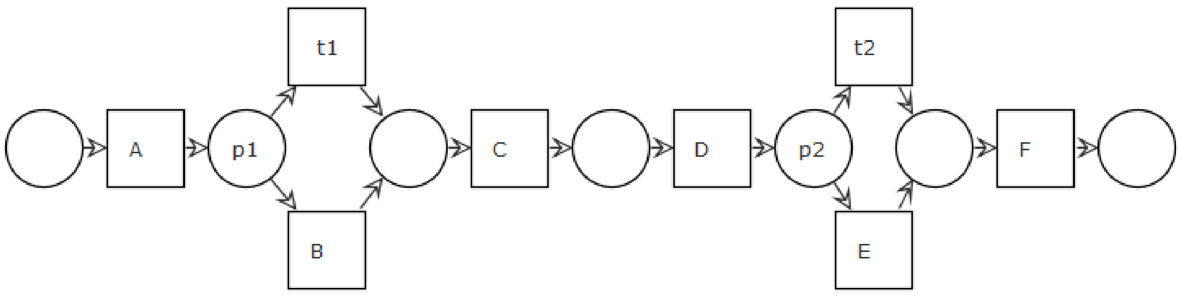


Figure 4.6. Example of a sound WF-net with invisibles tasks t1 and t2 not involved in a free choice construct (L = {<A, B, C, D, E, F>, <A, C, D, F>, <A, B, C, D, F>, <A, C, D, E, F>})

Definition 4.19 (*Improved reachable dependency related to invisible tasks*)

Let A be a set of activities, L be an event log over A, me be an activity from A, x,y be two invisible tasks, the improved reachable dependencies related to invisible tasks in a non-free choice are defined as follows:

- $x \succ_L m \Leftrightarrow \exists_{(a=\perp \vee a \in L) \wedge b \in L} (a \rightarrow_L x) \wedge (x \rightarrow_L b) \wedge b \succ_{L,pre=a} m \wedge$
  $\boldsymbol{\nexists_{n \in \sigma} b \succ_{L,pre=a} n}$,
- $m \succ_L x \Leftrightarrow \exists_{a \in L \wedge (b \in L \vee b=\top)} (a \rightarrow_L x) \wedge (x \rightarrow_L b) \wedge m \succ_{L,pre=b} a \wedge$
  $\boldsymbol{\nexists_{n \in \sigma} m \succ_{L,pre=n} a}$,
- $x \succ_L y \Leftrightarrow \exists_{(a_1=\perp \vee a_1 \in L) \wedge b_1 \in L \wedge a_2 \in L \wedge (b_2 \in L \vee b_2=\top)} (a_1 \rightarrow_L x) \wedge (x \rightarrow_L b_1)$
  $\wedge (a_2 \rightarrow_L y) \wedge (y \rightarrow_L b_2) \wedge b_1 \succ_{L,pre=a_1,post=b_2} a_2 \wedge$
  $\boldsymbol{\nexists_{n \in L} b_1 \succ_{L,pre=a_1,post=b_n} a_2}$.

4.2.3. Knowledge data base construction

All existing process discovery algorithms have no problem in correctly discovering the standard (sequence, choice, and parallel) constructs. The difference between the algorithms appears in the mining of complex constructs. Knowledge data-base will contain the information on the ability of each algorithm in mining the complex constructs. The process discovery techniques that are included in the knowledge data-base are α algorithm, α^{+}algorithm, α^{++}algorithm, $\alpha^{\#}$ algorithm, Inductive Miner (IM), HeuristicsMiner (HM), ILP (Werf et al.,2009), ETM (Buijs et al., 2012), Region Miner (RM), Transition System (TS) (Kalenkova et al, 2014), DWS (Greco et al, 2006), and Genetic Miner (GM). There are other algorithms but there are not implemented in ProM. Only the information of those implemented in ProM is included in the knowledge data-base. However, any newly developed algorithm and implemented in ProM can be included in the knowledge data-base. Knowledge data-base includes the information on the capability of any of these algorithms in discovering in detail the complex constructs: Short loops of length one (L_{1p} for short), Short loops of length two (L_{2p}), invisible tasks of type Short-Skip in sequence ($\mathrm{IvT}-\mathrm{SS}_{seq}$), invisible tasks of type Short-Skip in parallel ($\mathrm{IvT}-\mathrm{SS}_{par}$), invisible tasks of type Short-Redo in sequence ($\mathrm{IvT}-\mathrm{SR}_{seq}$), invisible tasks of type Short-Redo in parallel ($\mathrm{IvT}-\mathrm{SR}_{par}$), invisible tasks of type Long-Skip in sequence ($\mathrm{IvT}-\mathrm{LS}_{seq}$), invisible tasks of type Long-

Skip in parallel ($\mathrm{IvT} - \mathrm{LS}_{par}$), invisible tasks of type Long-Redo in sequence ($\mathrm{IvT} - \mathrm{LR}_{seq}$), invisible tasks of type Long-Redo in parallel ($\mathrm{IvT} - \mathrm{LR}_{par}$), non-free choice construct (NFC), and invisible tasks involved in a non-free choice construct (IvT-NFC).

(a) Algorithms classification based on their ability in mining complex constructs

There are plenty of empirical studies on the capability of process discovery techniques in mining complex constructs. However, most of studies focus on a small number of algorithms or on the most frequently used ones. Moreover, they do not evaluate the ability of algorithms in discovering all types of complex constructs. Therefore, we generated event logs containing all types of aforementioned complex constructs. We imported the event logs with the ProM tool and ran the plugin of each of the aforementioned algorithms.

α algorithm and α^{+} algorithm were not evaluated because we know before that α algorithm cannot discover the complex constructs and α^{+} algorithm can discover from complex constructs only the two types of short loops (L_{1p} and L_{2p}). We ran the experiments on the rest of algorithms. The results shows that α^{++} algorithm can correctly discover the two types of short loops and the three types of non-free choice constructs while it is incapable of discovering all types of invisible tasks and IvT-NFC. $\alpha^{\#}$ algorithm is able of mining L_{1p}, L_{2p}, $\mathrm{IvT} - \mathrm{SS}_{seq}$, $\mathrm{IvT} - \mathrm{SR}_{seq}$, $\mathrm{IvT} - \mathrm{LS}_{seq}$, $\mathrm{IvT} - \mathrm{LR}_{seq}$ whereas it is unable of discovering $\mathrm{IvT} - \mathrm{SS}_{par}$, $\mathrm{IvT} - \mathrm{SR}_{par}$, $\mathrm{IvT} - \mathrm{LS}_{par}$, $\mathrm{IvT} - \mathrm{LR}_{par}$, $\mathrm{IvT} - \mathrm{SW}$, NFC, and $\mathrm{IvT} - \mathrm{NFC}$. Inductive miner can discover correctly the two types of short loops, seven types of invisible tasks. However, this latter cannot detect one type of invisible tasks which are of type Switch and of type Short Redo in parallel, non-free choice constructs and invisible tasks involved in a non-free choice construct. Heuristic miner is capable of mining invisible tasks of types $\mathrm{IvT} - \mathrm{LR}_{seq}$, $\mathrm{IvT} - \mathrm{SS}_{seq}$, $\mathrm{IvT} - \mathrm{LS}_{seq}$, $\mathrm{IvT} - \mathrm{LS}_{par}$, $\mathrm{IvT} - \mathrm{SW}$, a similar behaviour to short loops of type L_{2p}, but it cannot discover short loops of type L_{1p}, invisible tasks of types $\mathrm{IvT} - \mathrm{SR}_{seq}$, $\mathrm{IvT} - \mathrm{SR}_{par}$, $\mathrm{IvT} - \mathrm{SS}_{par}$, non-free choice constructs, and invisible tasks involved in a non-free choice construct. Integer linear programming miner (ILP) can mine the two types of short loops, whereas it cannot discover all types of invisible tasks, non-free choice constructs, and invisible tasks involved in a non-free choice construct. Evolutionary Tree Mine (ETM) is capable of discovering short loops of types L_{1p}, invisible tasks of types $\mathrm{IvT} - \mathrm{SS}_{seq}$, $\mathrm{IvT} - \mathrm{LS}_{seq}$ while it is incapable of mining short loops of type L_{2p}, invisible tasks of types $\mathrm{IvT} - \mathrm{SR}_{seq}$, $\mathrm{IvT} - \mathrm{SR}_{par}$, $\mathrm{IvT} - \mathrm{LR}_{seq}$, $\mathrm{IvT} - \mathrm{LR}_{par}$, $\mathrm{IvT} - \mathrm{SS}_{par}$, $\mathrm{IvT} -$

LS_{par}, $\text{IvT} - \text{SW}$, non-free choice constructs, and invisible tasks involved in a non-free choice construct. Region miner (RM) is able of deriving short loops of type L_{2p}, invisible tasks of type similar behaviour to $\text{IvT} - \text{LR}_{seq}$, while it cannot discover short loops of types L_{1p}, the rest of invisible tasks, non-free choice constructs, and $\text{IvT} - \text{NFC}$. Transition system (TS) can derive short loops of types L_{2p} and similar behaviour to L_{1p}, invisible tasks of types $\text{IvT} - \text{SS}_{seq}$, similar behaviour to $\text{IvT} - \text{SR}_{seq}$, similar behaviour to $\text{IvT} - \text{SS}_{par}$. However, this technique cannot discover $\text{IvT} - \text{LR}_{seq}$, $\text{IvT} - \text{LR}_{par}$, $\text{IvT} - \text{LS}_{seq}$, $\text{IvT} - \text{LS}_{par}$, non-free choice constructs, and invisible tasks involved in a non-free choice structure. Disjunctive Workflow Schema (DWS) algorithm can mine short loops of type, invisible tasks of types $\text{IvT} - \text{SS}_{seq}$, $\text{IvT} - \text{SR}_{seq}$, $\text{IvT} - \text{LS}_{seq}$, $\text{IvT} - \text{LR}_{seq}$, $\text{IvT} - \text{SW}$. Nevertheless, it cannot derive short loops of types L_{1p}, all invisible tasks that are involved in a parallel construct, non-free choice constructs, and invisible tasks involved in a non-free choice construct. Finally, genetic miner (GM) can discover all constructs except invisible tasks of type $\text{IvT} - \text{SS}_{par}$, non-free choice constructs, and $\text{IvT} - \text{NFC}$. The abilities of these algorithms in mining the complex constructs are summarized in Table 4.1.

Table 4.1. Comparison of the ability of current algorithms in mining standard and complex constructs

	α^{++}	$\alpha^{\#}$	**IM**	**HM**	**ILP**	**ETM**	**RM**	**TS**	**DWS**	**GM**
$\mathbf{L_{1p}}$	Yes	Yes	Sb	No	Yes	Yes	No	Sb	No	Yes
$\mathbf{L_{2p}}$	Yes	Yes	Yes	Sb	Yes	No	Yes	Yes	Yes	Yes
$\mathbf{IvT_{SR_{seq}}}$	Sb	Yes	Yes	No	No	No	No	Sb	Yes	No
$\mathbf{IvT_{LR_{seq}}}$	No	Yes	Yes	Yes	No	No	Sb	No	Yes	Yes
$\text{IvT}_{\text{SR}_{\text{par}}}$	No	No	No	No	No	No	No	No	No	No
$\text{IvT}_{\text{LR}_{\text{par}}}$	No	No	Yes	No	No	No	No	No	No	Yes
$\mathbf{IvT_{SS_{seq}}}$	No	Yes	Yes	Yes	No	Yes	No	Yes	Yes	Yes
$\mathbf{IvT_{SS_{par}}}$	No	No	Yes	No	No	No	No	Sb	No	No
$\mathbf{IvT_{LS_{seq}}}$	No	Yes	Yes	Yes	No	Yes	No	No	Yes	Yes
$\mathbf{IvT_{LS_{par}}}$	No	No	Yes	Yes	No	No	No	No	No	Yes
$\mathbf{IvT_{SW}}$	No	Yes	No	Yes	No	No	No	No	Yes	Yes
NFC	Yes	No	No	No	No	No	No	No	No	Yes

$\boldsymbol{IvT_{in-NFC}}$	No	No	No	No	No	No	No	No	No	No

There is no discovery algorithm capable of discovering invisible tasks of type Short Redo in parallel and invisible tasks involved in a non-free choice constructs. Only $\alpha^{\$}$ algorithm can discover these two types of invisible tasks. However, this algorithm is not implemented in ProM. Therefore, we exclude the detection of invisible tasks of types $\mathrm{IvT}-\mathrm{SR}_{\mathrm{par}}$ and $\mathrm{IvT}-\mathrm{NFC}$.

(b) Mining time based classification of discovery algorithms

Process model discovery time may differentiate from algorithm to algorithm. Most of existing algorithms may take milliseconds to few minutes based on the size of the event logs to discover a process model. However, there are some algorithms which can take a long time to derive a process model such as genetic miner. This latter takes a long time in mining a process model even with a small event log and for big event log, it may run forever. Therefore, genetic algorithm might be the last candidate process discovery algorithm to recommend. The Evolutionary tree miner and Integer linear programing algorithms also take two much time to discover a model. Genetic miner takes time more than ETM and ETM takes time more than the ILP algorithm, and these three algorithms take time compared to the rest of algorithms.

(c) Algorithms classification based on their ability in discovering sound models.

There are process discovery algorithms that are not guaranteed to discover sound models. α algorithm series and heuristic miner might produce process models that are not sound. They may contain problems such as deadlocks, livelocks, etc. The inductive miner and ILP miner can guarantee soundness of discovered models (Jouck et al., 2018). A process model that is not sound cannot replay traces until the end.

Conclusion

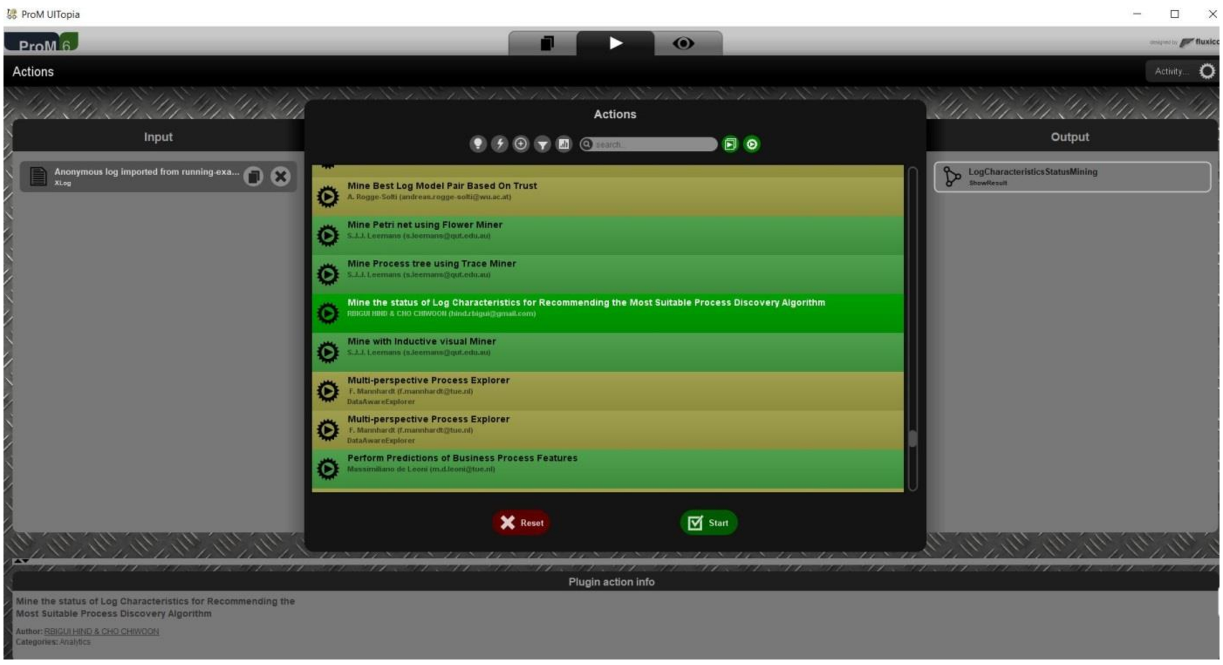
In this chapter, we proposed a framework for recommending the suitable process discovery algorithm to a given event log. The proposed methodology consists of detecting the complex control flow constructs existing in event log without discovering any model. Then recommending the best algorithm to a given event log based on a knowledge database containing information of the ability of existing algorithms on mining complex constructs, based on computation time, and based on the ability of model on discovering sounds models. The complex control-flow constructs detection is based on the relations introduced in Alpha algorithms (α^{++}, $\alpha^{\#}$, $\alpha^{\$}$), but in this work, the relations are used and improved to detect complex constructs from event logs without discovering any process model.

Chapter 5 RECOMMENDATION FRAMEWORK IMPLEMENTATION AND EVALUATION

This chapter discusses the experimental evaluation of the framework proposed for recommending business process discovery algorithms. First, we discuss the implementation in ProM, followed by an explanation of method used for evaluation. Then in section 5.3, we describe an evaluation based on 40 artificial examples. In section 5.4, we discuss an evaluation based on 5 more realistic event logs.

5.1. Implementation in ProM

As we have shown in the previous chapter, the framework consists of two parts. The first part is the detection of complex constructs from a given event log. The second part consists of recommending, based on knowledge data base, the algorithm capable of handling the constructs detected from the event log in the first part. The first part (i.e., the major part) which consists of detecting the complex characteristics (i.e., invisible tasks, non-free choice, loops) has been implemented in ProM. ProM is an extensible framework that provides a comprehensive set of plugins for the discovery, conformance checking and analysis of process models from event logs and can be downloaded from http://www.processmining.org. ProM takes an event log as input in the standard XES format and uses process mining plugins for different purposes. Knowledge in data base on the ability of algorithms in mining the complex constructs can be changed and updated with the conducted researches on enhancing the ability of existing algorithms in handling complex constructs. Figure 5.1 shows a screenshot of the plugin of detecting the status of complex constructs from a given event log.



Anonymous log imported from running-exa...
XLog

Mine Best Log Model Pair Based On Trust
A. Rogge-Solti (andreas.rogge-solti@wu.ac.at)

Mine Petri net using Flower Miner
S.J.J. Leemans (s.leemans@qut.edu.au)

Mine Process tree using Trace Miner
S.J.J. Leemans (s.leemans@qut.edu.au)

Mine the status of Log Characteristics for Recommending the Most Suitable Process Discovery Algorithm
RBIGUI HIND & CHO CHIWOON (hind.rbigui@gmail.com)

Mine with Inductive visual Miner
S.J.J. Leemans (s.leemans@qut.edu.au)

Multi-perspective Process Explorer
F. Mannhardt (f.mannhardt@tue.nl)
DataAwareExplorer

Multi-perspective Process Explorer
F. Mannhardt (f.mannhardt@tue.nl)
DataAwareExplorer

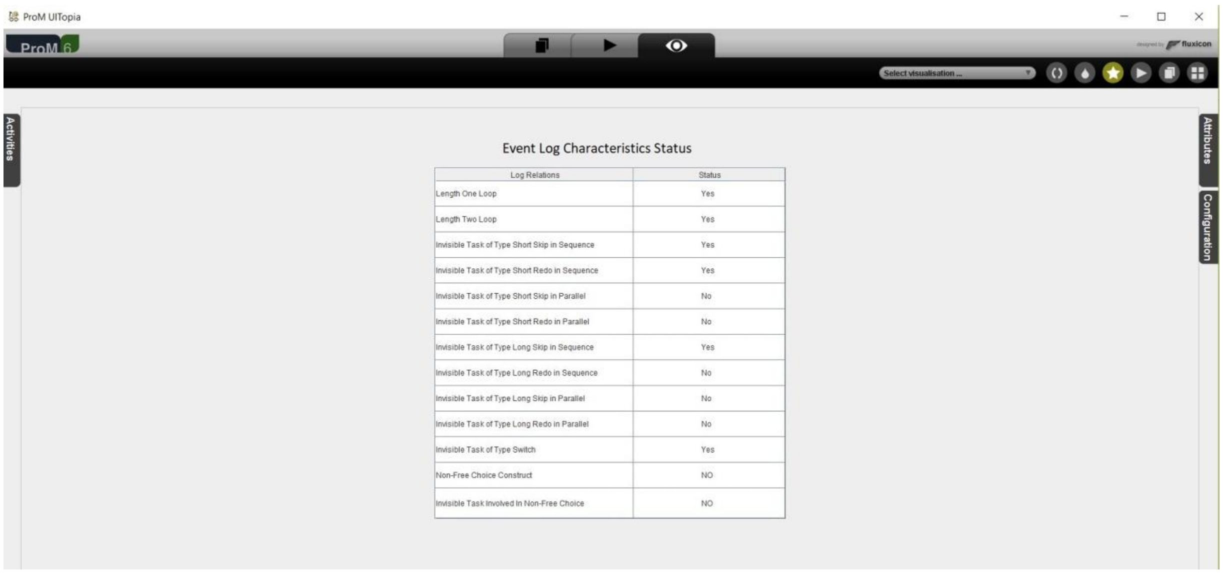
Perform Predictions of Business Process Features
Massimiliano de Leoni (m.d.leoni@tue.nl)

LogCharacteristicsStatusMining
ShowResult

Mine the status of Log Characteristics for Recommending the Most Suitable Process Discovery Algorithm
Author: RBIGUI HIND & CHO CHIWOON
Categories: Analytics

Figure 5.1. A screenshot of the implementation of the plugin detecting the status of complex constructs from event log

The plugin works as follows. Once the event log in question is imported in ProM, the plugin "Event Log Characteristics Detection for Recommending Process Discovery Algorithms" is selected and run. The output is a table which indicates whether the complex constructs (i.e., Short loops of length one (L_{1p} for short), Short loops of length two (L_{2p}), invisible tasks of type Short-Skip in sequence ($\mathrm{IvT} - \mathrm{SS}_{seq}$), invisible tasks of type Short-Skip in parallel ($\mathrm{IvT} - \mathrm{SS}_{par}$), invisible tasks of type Short-Redo in sequence ($\mathrm{IvT} - \mathrm{SR}_{seq}$), invisible tasks of type Short-Redo in parallel ($\mathrm{IvT} - \mathrm{SR}_{par}$), invisible tasks of type Long-Skip in sequence ($\mathrm{IvT} - \mathrm{LS}_{seq}$), invisible tasks of type Long-Skip in parallel ($\mathrm{IvT} - \mathrm{LS}_{par}$), invisible tasks of type Long-Redo in sequence ($\mathrm{IvT} - \mathrm{LR}_{seq}$), invisible tasks of type Long-Redo in parallel ($\mathrm{IvT} - \mathrm{LR}_{par}$), non-free choice construct (NFC), and invisible tasks involved in a non-free choice construct (IvT-NFC)) exist or not in the event log by showing "YES" if exist and "NO" if not. Then, based on the result obtained in the table and knowledge database, candidate algorithms are recommended. A screenshot of an example of obtained results is depicted in Figure 5.2.



Event Log Characteristics Status

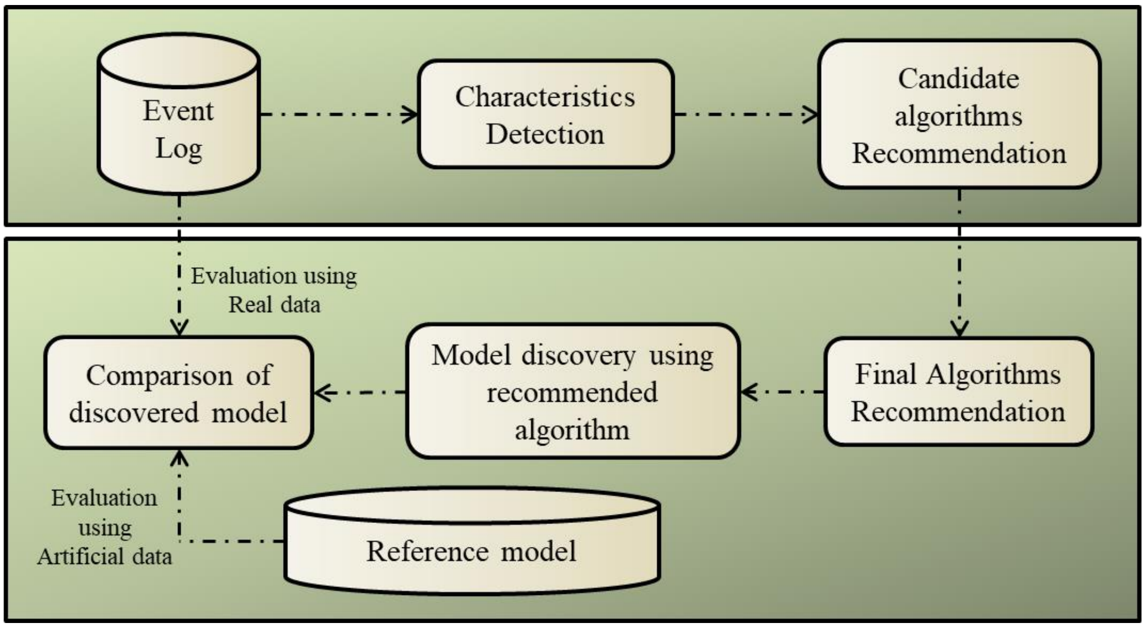
Log Relations	Status
Length One Loop	Yes
Length Two Loop	Yes
Invisible Task of Type Short Skip in Sequence	Yes
Invisible Task of Type Short Redo in Sequence	Yes
Invisible Task of Type Short Skip in Parallel	No
Invisible Task of Type Short Redo in Parallel	No
Invisible Task of Type Long Skip in Sequence	Yes
Invisible Task of Type Long Redo in Sequence	No
Invisible Task of Type Long Skip in Parallel	No
Invisible Task of Type Long Redo in Parallel	No
Invisible Task of Type Switch	Yes
Non-Free Choice Construct	NO
Invisible Task Involved In Non-Free Choice	NO

Figure 5.2. A screenshot of the result obtained from an example event log

5.2. Evaluation Framework

The framework has been evaluated using 40 artificial event logs and 5 real-life event logs. The 40 artificial event logs contain randomly the 11 characteristics $\mathrm{L_{1p}}$, $\mathrm{L_{2p}}$, $\mathrm{IvT-SS}_{seq}$, $\mathrm{IvT-SR}_{seq}$, $\mathrm{IvT-LS}_{seq}$, $\mathrm{IvT-LR}_{seq}$, $\mathrm{IvT-SS}_{par}$, $\mathrm{IvT-SR}_{par}$, $\mathrm{IvT-LS}_{par}$, $\mathrm{IvT-LR}_{par}$, and NFC. The characteristics existing in the logs were detected and candidate algorithms were recommended based on the detected characteristics and based on knowledge database containing information on the ability of α^{+}algorithm, α^{++}algorithm, $\alpha^{\#}$ algorithm, Inductive Miner (IM), Heuristics Miner (HM), Integer Linear Programming algorithm (ILP), Evolutionary Tree Miner (ETM), Region Miner (RM), Transition System (TS), and DWS on mining the complex constructs. If more than one algorithm is recommended, a recommendation based on time classification and soundness classification is conducted. Then, the final recommended algorithm is used to discover a process model from the event log in question. After that, for artificial event logs, the process model discovered with the final recommended algorithm is compared with the reference model of these event logs. In the practical world, usually reference models are not available. Therefore, for real life data, the model discovered with the recommended algorithm is replayed by the event log to investigate whether the event log is in conformance with the discovered process model. An illustration of the evaluation method is depicted in Figure 5.3.The comparison of the discovered model with the original model and the event log is conducted using conformance checking metrics. We used three types of metrics: behavioural similarity (B_P and B_R), to evaluate how similar the discovered model and the original model behaved in terms of precision and recall; and structural similarity ($S_P \ and \ S_R$), to assess how structurally similar the discovered model and the original model were in terms of precision and recall; ETC precision to identify whether the process model is precise to the observed behaviour (i.e., the event log). ETC precision is used when the reference model is unavailable. Behavioural and structural similarity metrics are used in the case of evaluation using artificial data and ETC precision metric is used in the case of evaluation using real-life data.

To identify similarity between two process models, an original model and a discovered model, the behavioural and structural similarity between them must be considered (de Medeiros et al., 2007). The behavioural similarity metrics measure the similarity in behaviour between two models in terms of precision and recall (de Medeiros et al., 2007). These metrics investigate the event log to quantify how similar the behaviour of discovered model is to that of its original model. This is done by replaying each trace against the two models and calculating how many activities are enabled in each model at the occurrence of every event in the trace. The more enabled activities the two models have in common, the higher is the similarity between them.



Event Log
Characteristics Detection
Candidate algorithms Recommendation
Evaluation using Real data
Comparison of discovered model
Model discovery using recommended algorithm
Final Algorithms Recommendation
Evaluation using Artificial data
Reference model

Figure 5.3. A screenshot of the result obtained from an example event log

Definition 5.1 (Behavioral precision and recall) (de Medeiros et al., 2007).

Some parameters are defined as follows:

σ: a trace in an event log.

L(σ): the number of occurrences of σ in an event log.

N_o *and* N_d*: the respective Petri nets for the original and the discovered models.*

C_o *and* C_d*: the respective causality relations for* N_o *and* N_d*.*

The behavioral precision and recall are defined as:

$$B_p(L, C_o, C_d) = (\sum (\frac{L(\sigma)}{|\sigma|} \times \sum_{i=0}^{|\sigma|-1} \frac{Enabled\ (C_o, \sigma, i)| \cap Enabled\ (C_d, \sigma, i)|}{|Enabled\ (C_d, \sigma, i)|})/\sum_{\sigma \in L} L(\sigma)$$

$$B_R(L, C_o, C_d) = (\sum (\frac{L(\sigma)}{|\sigma|} \times \sum_{i=0}^{|\sigma|-1} \frac{Enabled\ (C_o, \sigma, i)| \cap Enabled\ (C_d, \sigma, i)|}{|Enabled\ (C_o, \sigma, i)|})/\sum_{\sigma \in L} L(\sigma)$$

Where, $Enabled\ (C_o, \sigma, i)$ is the *set* of enabled activities when parsing the next event (or task) after position *i* in trace σ (de Medeiros et al., 2007). The value of both behavioral precision and recall metrics lies in the [0, 1] range. A value close to 1 indicates very high degree of similarity between the two models. The behavioral precision reflects how much of the behavior of the *discovered model* is also in the *original model*. The behavioral recall reflects how much of the behavior of the *original model* also occurs in the *discovered model*.

For structural similarity, the structural precision and recall metrics (de Medeiros et al., 2007) are used. The structural recall reflects the number of correct causality relations present in the *discovered model* as a fraction of the total number of causality relations in the *original model*. The structural precision reflects the fraction of correct causality relations present in the *discovered model*.

Definition 5.2 (Structural precision and recall) (de Medeiros et al., 2007).

Let NO = (Po, To, Fo) and Nd = (Pd, Td, Fd) be respective Petri nets for the original and discovered models. Let Co and Cd be the respective causality relations for No and Nd. The structural precision and structural recall are defined as:

$$S_p(N_o, N_d) = \frac{|C_o \cap C_d|}{C_d}$$

$$S_R(N_o, N_d) = \frac{|C_o \cap C_d|}{C_o}$$

Both structural precision and recall are in the range [0, 1]. A value close to 1 means they are very similar structurally.

5.3. Evaluation using artificial event logs

We have evaluated our approach using 40 artificial examples and compared the results of the proposed framework with empirical evaluation. The corresponding complete logs are generated manually. In the set of 40 models, the maximum number of activities in a process model is less than 15 and the number of cases in one event log is less than 30. The 40 reference models contain randomly the following characteristics $\mathrm{L_{1p}}$, $\mathrm{L_{2p}}$, $\mathrm{IvT-SS}_{seq}$, $\mathrm{IvT-SR}_{seq}$, $\mathrm{IvT-LS}_{seq}$, $\mathrm{IvT-LR}_{seq}$, $\mathrm{IvT-SS}_{par}$, $\mathrm{IvT-SR}_{par}$, $\mathrm{IvT-LS}_{par}$, $\mathrm{IvT-LR}_{par}$, and NFC. For this analysis, we used visual inspection (the mined model is similar to reference model) and the conformance metrics aforementioned before i.e., B_r (behavioural recall), B_p (behavioural precision), S_r (structural recall), and S_p (structural precision).

By visually comparing the reference models and the model discovered by the recommended algorithm, we find that 95% of process models (38 models out of 40) are similar to reference models. In addition to the visual inspection, the conformance metrics for testing behavioural and structural similarities between originals models and the models discovered by recommended algorithms are computed. The obtained results are illustrated in Figure 5.3. As can be seen, the results obtained by computing conformance metrics are similar to the results obtained by visual inspection. 38 process models are looks similar to the original models (the values are equal or close to 1). Only two discovered models are not similar to the reference models. These are discovered from the events logs L29 and L38. For L29, structural similarity metrics are slightly far from value 1 but the behavioural similarity metrics are close to 1. If we look at the original model, it contains a loop of length one while in the discovered model, there is an invisible task of type short redo in sequence. Actually, these two structures are different but they have similar behaviour. This justifies the values of structural similarity metrics being far from 1. For L38, the original model contains two invisible tasks involved in a non-free choice construct. However, in the discovered model, the two invisible tasks are correctly discovered and the implicit dependency between the two invisible tasks which reflects the construct of a non-free choice is not discovered. This is also logical, since none of the existing algorithms implemented in ProM is capable of discovering invisible tasks involved in a non-free choice construct.

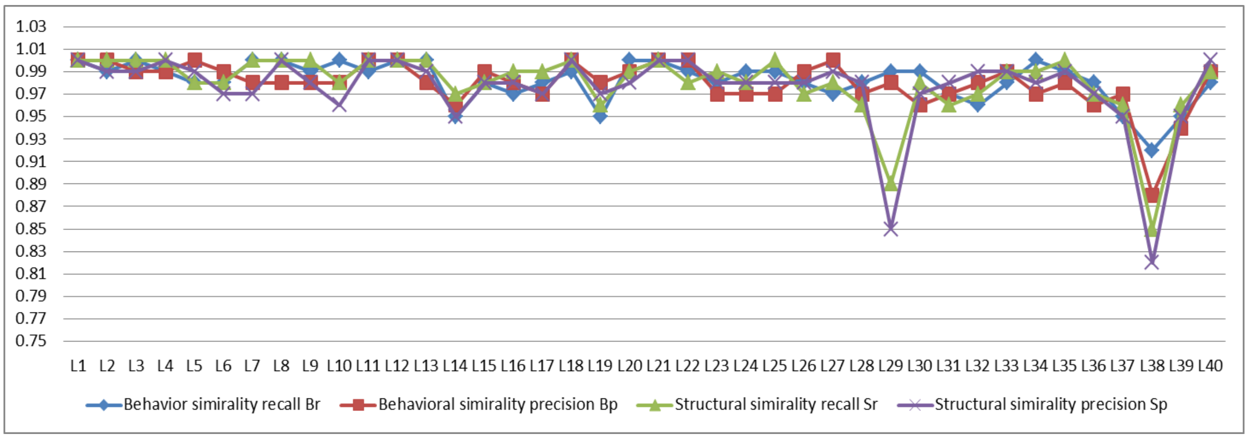


Figure 5.4. Comparison of mining results by recommended algorithms with reference models on complete logs generated from artificial examples.

5.4. **Evaluation based on real-life event logs**

The evaluation using artificial event logs shows that the proposed framework is powerful enough to recommend a process discovery algorithm suitable to a given event log by detecting complex constructs in the event log. Now we use more realistic examples to show the applicability of the proposed framework. We conducted the evaluation using 5 real-life event logs taken from the Process Mining Manifesto database and from the Shipbuilding Processing Plan Management System of a heavy manufacturing company. First, we detected the existing constructs from the event logs, and then we recommended the suitable algorithms using the constructed knowledge database. The obtained results are illustrated in Table 5.1. After that, we discovered for each event log, process models using different process discovery algorithms. Then, to evaluate the discovered models, we checked the conformance between the mined models and the corresponding event logs using the plugin of ProM "*Check Conformance using ETConformance*". This conformance checker provides information on how much the mined model is precise to the corresponding event log. The precision metric is called ETC precision. The obtained results are shown in Table 5.2. Then, we compare the the of precision metric of each discovered model by each algorithm and for each event log recommended algorithm in Table 5.2 with recommended algorithm in Table 5.1. The models were discovered using Inductive Miner (IM), Heuristics Miner (HM), Alpha++ algorithm, Alpha # algorithm, Integer Linear Programming Miner (ILP), Evolutionary Tree Miner (ETM), DWS algorithm, Genetic Miner and Region Miner. Only IM, HM, ILP, DWA, Alpha#, Alpha++ algorithms are shown in Table 5.2. The others are not illustrated due to the running time which exceeded five minutes.

Table 5.1. Recommended algorithm for each event log

Event logs	**L1**	**L2**	**L3**	**L4**	**L5**
#cases	97	110	663	1676	31509
#events	361	3770	19658	129428	1202267
#tasks	8	21	18	38	66
Recommended algorithm	Alpha^{++}	IM/ILP	IM	IM/DWS	ILP

Table 5.2. The ETC precision values of discovered models

	HM	**Alpha++**	**IM**	**Alpha#**	**ILP**	**DWS**
L1	0.7973	1.0	0.8	0.8	0.8	0.69
L2	0.3811	0.6665	0.6665	0.6665	0.7	0.6665
L3	-	-	0.945	0.4423	0.361	0.3
L4	0.4	0.03	0.8709	0.02	-	0.89
L5	0.681	-	0.89	-	0.91	0.45

(-) The value could not be computed.

As can be seen in Table 5.1, Alpha++ was recommended to be a suitable process discovery algorithm for the event log L1 due to the detection of a non-free choice construct in the event log L1 and due to the fact that the numbers of events, cases and tasks are not that big. In Table 5.2, the precision value of the model discovered by Alpha++ algorithm is the higher among the others (equal to 1). The precisions values of the models mined by IM, HM, ILP, Alpha#, DWS is lower. This is logical since those algorithms are incapable of handling non-free choice constructs. For log L2, no complex construct was detected. Thus, any one of the five algorithms can be recommended. However, since Alpha algorithms and Heuristic Miner do not guarantee sound models with big event logs, we recommended IM, ILP and DWS. The values of precision of IM and DWS are similar and the precision value of ILP is a slightly higher. For log L3, invisible task of type short skip in sequence was detected. Therefore, IM algorithm was recommended. In Table 5.2, the precision value of IM was the highest. The results obtained for logs L4 and L5 are similar to the previous ones. Based on the obtained results, we can conclude that the recommendation framework is working well.

Conclusion

In this chapter, we described the implementation of the proposed framework in ProM. Then, we discussed the methodology and metrics used for the evaluation. The framework has been evaluated using artificial data and real-life data. The obtained results show that the recommended algorithms are correctly recommended. However, the framework has been evaluated with small number of real-life event logs. In the future work, we will use more real-life data.

Chapter 6 HEURISTIC RULE-BASED ALGORITHM

Several process discovery algorithms have been developed and applied successfully. However, there are complex constructs that current discovery techniques cannot correctly discover in models based on event logs. These complex constructs are short loops, invisible tasks, duplicate tasks, and non-free choice constructs. There is currently no algorithm that can handle all of these structures in a restricted time. For instance, the Inductive Miner algorithm (Leemans *et al.*, 2013) is robust for invisible tasks, but it cannot handle duplicate tasks and non-free choice constructs. Another example is the Region-based algorithm (Bergenthum et al., 2007), which is capable of mining some non-free choice constructs but cannot handle invisible and duplicate tasks. The other algorithms have similar problems. Therefore, the aim of this chapter is to introduce a novel algorithm that is capable of constructing a workflow model with short loops, invisible tasks, non-free choice constructs, and duplicate tasks. This technique is called the Heuristic Rule-based algorithm (i.e., HR).

The Heuristic Rule-based algorithm focuses on the control-flow perspective of process mining by designing process models based on a given event log. The starting point of the HR algorithm is the construction of the Ancestor/Descendent Table for each activity in each event trace in the event log. The Ancestor/Descendent Table (A/D Table) is similar to the Predecessor/Successor Table (P/S Table). The idea of constructing a P/S table was first introduced in (Li *et al.*, 2007) to extend the $\alpha -$Algorithm to duplicate task mining. Here, the A/D table is used as a starting point of a new algorithm that mines all the standard and complex characteristics that might exist in an event log, such as sequences, parallelism, exclusive choices, invisible tasks, duplicate tasks, short loops, and non-free choice constructs. First, we introduce the A/D Table, the starting point of our approach. Second, we explain how to detect the standard constructs. Third, we provide the heuristic rules that can detect the complex constructs. Finally, we present the algorithm. In this paper, we assume: (i) the event log is complete (a task is following another task directly iff the log contains an example of this behaviour), and (ii) there is no noise in the log (each event recorded in the event log is correct and is representative of the behaviour of the model to be discovered). Related work is discussed in Chapter 1.

6.1. Elaboration of the Ancestor/Descendant Table

The starting point of our Heuristic Rule-based algorithm is the construction of the A/D

Table for each activity in each trace. For each activity *X* that occurs in an event trace, the following information is extracted from the event log: (i) the name of the activity that directly precedes activity *X* (we use $T_a(X)$ to denote the ancestor of *X (Predecessor of X))*, and (ii) the name of the activity that directly follows activity *X* (we use $T_d(X)$ to denote the descendent of *X (Successor of X))*. The information extracted about activity *X* is gathered in the A/D table.

Consider, for example, the event log shown in Table 6.1. It consists of 10 activities, and only distinctive event traces that represent all possible occurrences of every activity are illustrated. Table 6.2 presents the ancestor and descendant of every activity B in each representative trace.

Table 6.1. Example of an event log (containing only possible event traces)

Case id	**Event Trace**
σ_1	*A B C E F G I J*
σ_2	*A B A B C E F G I J*
σ_3	*A B D E F H J*
σ_4	*A B A B D E F H J*

The activity identifier $\sigma_i(X,n)$ indicates the *n*th occurrence of an activity called *X* in the event trace σ_i. For instance, $\sigma_1(B,1)$ refers to the first occurrence of activity *B* in σ_1, and $\sigma_4(B,2)$ refers to the second occurrence of activity *B* in σ_4. The ancestor of the first occurrence of activity *B* in the event trace σ_1 is activity *A*. We denote $T_a(B,1)_{\sigma_1} = A$. The descendent of the second occurrence of activity *B* in the event trace σ_4 is activity *D*. We denote $T_d(B,2)_{\sigma_4} = D$.

Table 6.2. An example of an A/D table for activity B

Activity identifier	T_a	T_d
$\sigma_1(B,1)$	*A*	*C*
$\sigma_2(B,1)$	*A*	*A*
$\sigma_2(B,2)$	*A*	*C*
$\sigma_3(B,1)$	*A*	*D*
$\sigma_4(B,1)$	*A*	*A*
$\sigma_4(B,2)$	*A*	*D*

6.2. Identification of Standard Constructs

Any workflow net can be constructed from the standard constructs (sequence, AND-split/join, and XOR-split/join) to represent sequences, choices, and parallelism. A process model can be obtained from the event log by a search for causal dependencies (Van der Aalst *et al.*, 2006). For example, a task that always follows another task might indicate a causal dependency between the two tasks. The following notations are designed to detect such dependencies and mine the standard constructs based on the A/D Table. Let L be an event log and $\sigma_k \in L$ be the event traces.

1. $T_d(x) = \{y, z\}$ if there are $x, y, z \in \sigma_i, \sigma_j$ and $\sigma_i, \sigma_j \subseteq \sigma_k$ such that $\forall \mathrm{i} = \mathrm{j}$ or $\mathrm{i} \neq \mathrm{j}$: $T_d(x, n)_{\sigma_i} = y$ and $T_d(x, m)_{\sigma_j} = z$,
2. $T_a(x) = \{y, z\}$ if there are $x, y, z \in \sigma_i, \sigma_j$ and $\sigma_i, \sigma_j \subseteq \sigma_k$ such that $\forall \mathrm{i} = \mathrm{j}$ or $\mathrm{i} \neq \mathrm{j}$: $T_a(x, n)_{\sigma_i} = y$ and $T_a(x, m)_{\sigma_j} = z$,
3. $\boldsymbol{x>y_{i,i\geq 1}}$ if $\forall \mathrm{i} \geq 1$: $T_d(x) = \{y_i\}$ and $x \notin T_d(y_i)$,
4. $\boldsymbol{x//y}$ if there are $x, y \in \sigma_i, \sigma_j$ and $\sigma_i, \sigma_j \subseteq \sigma_k$ such that $\forall \mathrm{i} \neq \mathrm{j}$: $T_d(x, n)_{\sigma_i} = y$ and $T_d(y, m)_{\sigma_j} = x$, and $\nexists \sigma_l, \sigma_o$ such that $T_d(x)_{\sigma_l} = x$ and/or $T_d(y)_{\sigma_o} = y$,
5. $\boldsymbol{x<//(y,z)}$ if there are $x, y, z \in \sigma_k$ such that $x > y$ and $x > z$ and $y//z$,
6. $\boldsymbol{//(x,y)<z}$ if there are $x, y, z \in \sigma_k$ such that $x > z$ and $y > z$ and $x//y$,
7. $\boldsymbol{x<V(y,z)}$ if there are $x, y, z \in \sigma_k$ such that $x > y$ and $x > z$ and $y \notin T_d(z)$ and $z \notin T_d(y)$,
8. $\boldsymbol{V(x,y)<z}$ if there are $x, y, z \in \sigma_k$ such that $x > z$ and $y > z$ and $x \notin T_a(y)$ and $y \notin T_a(x)$.
9. $\boldsymbol{I:x}$ if there is $x \in \sigma_k$ such that $T_a(x)_{\sigma_k} = \{\emptyset\}$,
10. $\boldsymbol{x:O}$ if there is $x \in \sigma_k$ such that $T_d(x)_{\sigma_k} = \{\emptyset\}$.

If there are $x, y, \in \sigma_i$ such that: $T_d(x)_{\sigma_i} = y$, this indicates that the descendent of task x in σ_i is task y. Based on this, the ordering relations are extracted. In definition (1), $T_d(x) = \{y, z\}$ signifies that an activity x can be followed by y and by z. This can be achieved iff x has two different descendants in different event traces or if two different occurrences of x have distinct descendants in the same event trace. Note that a task can have one or multiple descendants; thus, one task can be followed directly by one or multiple tasks. Definition (2) is similar to (1). Definition (3) specifies that, if x can be directly followed by one or multiple tasks y_i, such that y_i is never followed by *x*, then there is a causal dependency between tasks *x* and y_i. This causal dependency is denoted by (>). Definition (4) indicates that if the descendent of task *x* is equal to *y* in trace σ_i and the descendent of task *y* is equal to *x* in a different trace, and there is no trace where the descendants of *x* and/or *y* are equivalent to *x* and *y* respectively, then the two tasks *x* and *y* are in parallel. The relation (<//) in definition (5) identifies the AND-split that allows more than one task to be executed simultaneously. *x<//(y,z)* indicates that two tasks *y* and *z* occur in parallel, and that their concurrency happens just after task *x* is executed. This can be achieved iff *x* is directly followed by *y* and z, and *y* and *z* are in parallel. In definition (6), the relation (//<) denotes the AND-join in order to synchronize concurrent workflows, and *//(x,y)<z* specifies that two parallel tasks *x* and *y* are joined, after their execution, in task *z*. The relations (<*V*) and (*V*<) in definitions (7) and (8) specify the XOR-split and the XOR-join, respectively, for an exclusive choice between two or more tasks. *x<V(y,z)* refers to the situation in which the choice between two tasks *y* and *z* is made after the execution of task *x*, whereas *V(x,y)<z* states that task *z* happens after only one of the tasks *x* and *y* is executed. One can write *x<//*$(y_1, y_2,..y_n)$ to indicate that several activities $y_1, y_2,..y_n$ occur in parallel, and the AND-split is in task *x*. Similarly, one can write *V(*$y_1, y_2,..y_n$*)<z* to indicate that an exclusive choice can be made from multiple tasks $y_1, y_2,..y_n$, and the XOR-join is in task *z*. Finally, the notation (*I:*) in definition (9) refers to a start activity, while the relation (:*O*) in definition (10) specifies an end activity.

In this section, the notations that can detect the standard constructs of a process model from a given event log on the basis of the A/D table are introduced. However, real-life processes contain complex constructs such as loops, invisible tasks, duplicate tasks, and non-free choice constructs. These complex characteristics are addressed in the following section.

6.3. Identification of Complex Constructs

In this section, advanced heuristic rules are introduced to detect invisible tasks, short loops, duplicate tasks, and non-free choice constructs.

6.3.1. Invisible Tasks Discovery

An invisible task is a hidden task that has been executed but cannot be recorded in the event log. Invisible tasks were classified for the first time by (Wen et al., 2007b) into four types according to the functional structure: REDO, SKIP, SIDE, and SWITCH. REDO-type invisible tasks can be divided into two subtypes: short and long. Similarly, SKIP-type invisible tasks can be split into short-skip and long-skip. SIDE-type invisible tasks can also be divided into two types: side-type at the beginning and side-type at the end. There is only one type of SWITCH. These seven types of invisible tasks will be detected with the following advanced heuristic rules.

(a) Detection of Invisible Tasks of Type Short Redo (IvT-SR)

An *IvT-SR* is a task that allows the execution of one task to be repeated. Consider the complete log L_1 and its process model, represented in a Petri net graph in Figure 6.1(a), which contains an invisible task *t* allowing the repetition of *b*. The starting point of our HR algorithm for any type of mining is the construction of the A/D Table. The A/D Table of L_1 is presented in Table 6.3. Accordingly, the following dependencies are extracted: $a>b$ and $b>c$. However, task b can be repeated several times, and the relation (>) cannot detect the repetition of task b because it does not allow a task to be followed by itself. Therefore, direct successions (i.e., $T_d(x) = y$) need to be used. Any repeatable task b can be determined by a search of the ordering bb in the event traces of the log. In other words, such a task can be identified by a search of the direct succession $T_d(b)_{\sigma_i} = b$ or $T_a(b)_{\sigma_i} = b$. Based on this observation, let *L* be an event log and $\sigma_i \in L$ be the event traces. We introduce the heuristic rule that can detect an *IvT-SR* as follows:

$$IF\ there\ exists\ x \in\ \sigma_i\ such\ that\ T_d(x)_{\sigma_i} = x\ OR\ T_a(x)_{\sigma_i} = x,\ THEN\ there\ exists\ t \in IvT - SR\ such\ that\ x > t\ \text{AND}\ t > x \quad \text{(R1)}$$

Rule (1) states that, if there exists a task *x* that belongs to a trace σ_i such that the descendent or ancestor of *x* is equivalent to *x*, then there exists an invisible task *t* that allows the repetition of *x*. The detected invisible task will be placed such that *t* follows *x* and is followed by *x*.

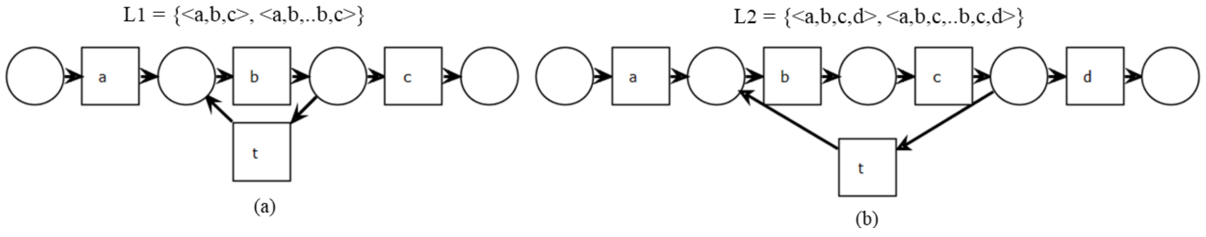

L1 = {<a,b,c>, <a,b,..b,c>}
a
b
c
t
(a)
L2 = {<a,b,c,d>, <a,b,c,..b,c,d>}
a
b
c
d
t
(b)

Figure 6.1. Sound process models of invisible tasks of type short redo (a) and long redo (b)

Table 6.3. The A/D Table of L1

Task id	$\sigma_1(a,1)$	$\sigma_2(a,1)$	$\sigma_1(b,1)$	$\sigma_2(b,1)$	$\sigma_2(b,k)$	$\sigma_1(c,1)$	$\sigma_2(c,1)$
$\boldsymbol{T_a}$	$\emptyset$	$\emptyset$	a	a	b	b	b
$\boldsymbol{T_d}$	b	b	c	b	c	$\emptyset$	$\emptyset$

(b) Detection of Invisible Tasks of Type Long Redo (IvT-LR).

An invisible task of type long redo is a task that allows repetition of two or more activities. Figure 6.1(b) depicts a Petri net model containing an invisible task of type long redo (*IvT-LR*) and its corresponding complete log L_2. The A/D Table of log L_2 is presented in Table 6.4. Accordingly, we obtain only *a>b* and *b>d*. Since $T_d(c,1)_{\sigma_2} = b$ and $T_d(b,1)_{\sigma_2} = c$, the relation (>) cannot be used to provide insight into tasks b and c. Therefore, the heuristic rule to detect invisible tasks of type long redo will be strongly based on direct succession relations. An IvT-LR can allow the repetition of several tasks. Let $x_1, x_2, \dots, x_n \in \sigma_j$ be the tasks to be repeated by an IvT-LR, and let i be the ith occurrence of $x_1, x_2, \dots, x_n$. If we have two repetitions, we will have a sequence of $x_1 x_2 \dots x_n x_1 x_2 \dots x_n$; if the repetition is performed three times, we will have $x_1 x_2 \dots x_n x_1 x_2 \dots x_n x_1 x_2 \dots x_n$; and so on. It is clear in the A/D Table that the descendent of the last occurrence of the last task x_n is different than x_1, while the other occurrences of x_n is equivalent to x_1. Let L be an event log and $\sigma_K \in L$ be the event traces. Based on these observations, the heuristic rule (2) to discover an *IvT-LR* is defined as follows.

$$IF\ \exists x_1, x_2, \dots, x_p \in \sigma_j\ AND\ \exists i \in \{1,..,q\}\ AND\ \forall k \in \{1,..,p-1\}\ such\ that\ T_d(x_k, i)_{\sigma_j} = x_{k+1}\ AND\ T_d(x_k, i+1)_{\sigma_j} = x_{k+1}\ AND\ for\ k{=}p \wedge i{\neq}q{:}\ T_d\big(x_p, i\big)_{\sigma_j} = x_1\ AND\ for\ k{=}p \wedge i{=}q{:}\ T_d\big(x_p, q\big)_{\sigma_j} \neq x_1\ THEN\ \exists t \in IvT-LR\ such\ that\ x_p > t\ AND\ t > x_1 \quad \text{(R2)}$$

Rule (2) states that if there exist p tasks x_1, x_2, ..., and x_p which can be repeated i times such that for 1≤k≤p-1, the descendants of task x_k are equal in the occurrences i and $i+1$, and for k=p (the last activity), the descendent of x_p is equivalent to x_1 in the occurrence i and different from x_1 in the occurrence $i+1$, then there is an invisible task of type long redo which can allow the repetition of x_1, x_2, ..., and x_p.

Table 6.4. The A/D Table of L2

Task id	$\sigma_1(a,1)$	$\sigma_2(a,1)$	$\sigma_1(b,1)$	$\sigma_2(b,1)$	$\sigma_2(b,k)$	$\sigma_1(c,1)$	$\sigma_2(c,1)$	$\sigma_2(c,k)$	$\sigma_1(d,1)$	$\sigma_2(d,1)$
$\boldsymbol{T_a}$	∅	∅	a	a	c	b	b	b	c	c
$\boldsymbol{T_d}$	b	b	c	c	c	d	b	d	∅	∅

(c) Detection of Invisible Tasks of Type Short Skip (IvT-SS).

Flexibility is one of the primary characteristics of modern execution systems. In many cases, skipping the execution of one or more tasks is required for rapid execution of a process. Skipping the execution of some activities is actually an action that has been taken; however, this action is not recorded in the information system, which makes it difficult for process miners to detect its execution based on the event log. The skipped execution of one task is represented by an invisible task called a short skip (IvT-SS). Consider the complete event log L_3 and its Petri net model depicted in Figure 6.2(a). Since there is no repetition of tasks in the log, the notation (>) can be used, as well as its derivatives. The A/D Table of log L_3 is presented in Table 6.5. Accordingly, we obtain the following causal dependencies: *a>b*, *a>c*, and *b>c*. Even though we have *a>b* and *a>c*, the conditions of AND-split and XOR-split are not satisfied. Similarly, for *a>c* and *b>c*, the conditions of AND-join and XOR-join are not satisfied. Since tasks *a, b,* and *c* are not involved in any choice or concurrency, it is clear from the causal dependencies *a>b*, *b>c*, and *a>c* that there is only one case, an invisible task that skips the execution of task *b* between task *a* and task *c* (*a>c*). The rule for discovering an *IvT-SS* is given below:

IF there are $x, y, z \in \sigma_k$ *such that* $x > y$ *AND* $y > z$ *AND* $x > z$ *AND not* $x <//(y,z)$ *AND not* $x < V(y,z)$ *AND not* $//(x,y) < z$ *AND not* $V(x,y) < z$, *THEN there exists* $t \in IvT-SS$ *such that* $x < V(y,t)$ *AND* $V(y,t) < z$ (R3)

Rule (3) determines that, if three tasks *x*, *y*, and *z* exist such that there are causal dependencies between *x* and *y*, between *y* and *z*, and between *x* and *z*, and these dependencies are not involved in an AND-split, XOR-split, AND-join, or XOR-join, then there exists an invisible task *t* that allows the skipping of *y*. This invisible task will be placed such that there is an exclusive choice between *t* and *y*.

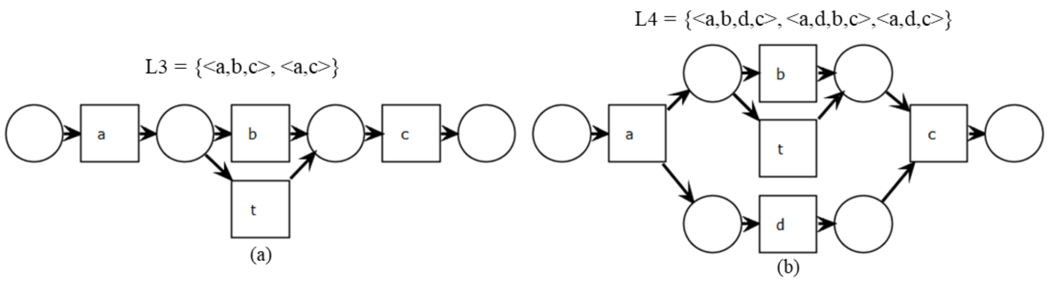


Figure 6.2. Short-skip invisible task in a sequential workflow (a) and a concurrent workflow (b)

Table 6.5. The A/D Table of L3

Task id	$\sigma_1(a,1)$	$\sigma_2(a,1)$	$\sigma_1(b,1)$	$\sigma_1(c,1)$	$\sigma_2(c,1)$
$\boldsymbol{T_a}$	∅	∅	a	b	a
$\boldsymbol{T_d}$	b	c	c	∅	∅

Rule (3) attempts to detect an invisible task of type short skip that is located in the middle of a workflow. Let us assume that the place after *b* and *t* in Figure 6.2 (a) is the sink place (the end place), and that task *c* does not exist, or the task before b and t is the source place (the start place) and that task *a* does not exist. Rule (3) will not be able to detect this case. Therefore, we define a rule that is a special case of rule (3):

IF there are $x, y, z \in \sigma_k$ *such that* $x > y$ *AND* $[y{:}\,O$ *AND* $x{:}\,O]$ *OR* $[I{:}\,y$ *AND* $I{:}\,x]$, *THEN there exists* $t \in IvT - SS$ *such that* $[x < V(t,y)$ *AND* $t{:}\,O]$ *OR* $[V(t,x) < y$ *AND* $I{:}\,t]$ (R4)

Rule (4) attempts to detect an invisible task of type skip that allows the execution of a task *y* to be skipped when *y* is directly connected either to the sink place or the source place. The invisible task *t* will be placed such that there is an exclusive choice between *t* and *y*, and *t* is directly connected to the sink place or there is an exclusive choice between *t* and *x*, and *t* is directly connected to the source place.

Now, rules (3) and (4) can correctly detect an IvT-SS, but only in a sequential workflow. In a concurrent workflow, the situation is different. Consider the complete event log L_4 depicted in Figure 6.2(b). The A/D Table of L_4 is shown in Table 6.6.

Table 6.6. The A/D Table of L_4

Task id	$\sigma_1(a,1$	$\sigma_2(a,1)$	$\sigma_3(a,1)$	$\sigma_1(b,1)$	$\sigma_2(b,1)$	$\sigma_1(c,1)$	$\sigma_2(c,1)$	$\sigma_3(c,1)$	$\sigma_1(d,1)$	$\sigma_2(d,1)$	$\sigma_3(d,1)$
T_a	∅	∅	∅	a	d	d	b	d	b	a	a
T_d	b	d	d	d	c	∅	∅	∅	c	b	c

Based on Table 6.6, we obtain the following dependencies: *a>b*, *a>d*, *b>c*, *d>c*, a*<//(b, d)*, and *//(b, d)<c*. These dependencies only refer to the existence of the concurrent tasks b and d, the AND-split in task a, and the AND-join in task c and do not provide any insight into the task that has been skipped. $\sigma_3 = < a, d, c >$ is the only trace that does not contain task *b*. In other words, only trace σ_3 can provide insight into the skipped task *b*, while the two remaining traces can only provide insight into the concurrency. Let L be a complete log and $\sigma_k \in L$ be the event traces. Accordingly, we define the heuristic rule that derives an *IvT-SS* in a concurrent workflow as follows:

$$IF\ \exists w, x_1, x_2, .., x_n, z \in \sigma_k\ such\ that\ w <//(x_1, x_2, .., x_n)\ AND\ //(x_1, x_2, .., x_n) < z\ AND$$
$$\exists \sigma_i \subseteq \sigma_k{:}\ T_a(x_{j/1 \le j \le n})_{\sigma_i} = w\ AND\ T_d(x_{j/1 \le j \le n})_{\sigma_i} = z,\ THEN\ \exists t \in IvT - SS_{par}\ such\ that$$
$$\mathrm{w} < V(x_1, x_2, .., x_n, t)\ AND\ V(x_1, x_2, .., x_n, t) < z\ \text{(R5)}$$

Rule (5) states that, if there exist concurrent tasks $x_1, x_2, ..$, and x_n, such that the ancestor of x_j is the activity splitting the AND of the concurrency and the descendent of x_j is the activity joining the AND of the concurrency, then there exists an invisible task that allows the execution of x_j to be skipped. The detected invisible task will be placed such that there is an exclusive choice between $x_1, x_2, .., x_n$ and *t.* For an invisible task of type short skip in a sequential workflow, we use the notation t $\in IvT - SS_{seq}$; for an invisible task of type short skip in a concurrent workflow, we use the notation t $\in IvT - SS_{par}$.

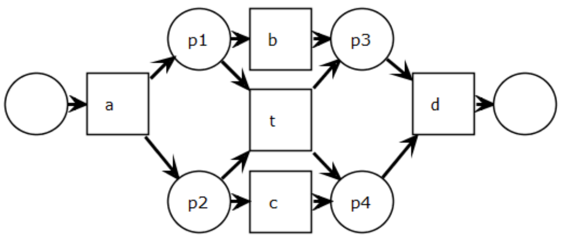


Figure 6.3. Process model with an invisible task of type skip combined with a parallel construct

There is another type of invisible tasks of type short skip in a parallel workflow. For instance, Figure 6.3 depicts a process model with an invisible task *t* combined with the parallel construct (b,c,p1,p2,p3,p4). This type of invisible tasks can be detected with the following rule:

$$IF\ \exists w, x_1, x_2, .., x_n, z \in \sigma_k\ such\ that\ w <//(x_1, x_2, .., x_n)\ AND\ //(x_1, x_2, .., x_n) < z\ AND\ w > z\ THEN\ \exists t \in IvT - SS_{par}: w < V(t, x_{j/1\leq j\leq n})\ \text{AND}\ V(t, x_{j/1\leq j\leq n}) < \text{z}\ \text{(R6)}$$

(d) Detection of Invisible Tasks of Type Long Skip (IvT-LS).

An invisible task of type long skip (*IvT-LS*) is a hidden task that allows two or more tasks to be skipped in the execution. Since the behaviour of an *IvT-LS* in a sequential workflow can differ from its behaviour in a concurrent workflow (similar to an *IvT-SS*), the two cases will be handled separately.

To derive an *IvT-LS* in a sequential workflow, consider the complete log L_5 and its process model, as depicted in Figure 6.4. The A/D Table of L_5 is presented in Table 6.7. The following dependencies are extracted from Table 8: *a>b*, *a>d*, *c>d*, *b>c*, *a<V(b,d)*, and *V(a,c)<d*. However, the last two dependencies are unrealistic because task *a* is splitting an XOR-split of two tasks *b* and *d* whereas *d* is joining an XOR-join of task a and another task. For this, we introduce definition 6.1.

Definition 6.1. *Let L be a complete event log and* $\sigma_k \in L$ *be the event traces.*

- *If there are* $w, x, y, z \in \sigma_k$ *such that* $w < V(x, z)$ *and* $V(w, y) < z$, *then* not $w < V(x, z)$ *and* not $V(w, y) < z$.
- *If there are* $w, x, y, z \in \sigma_k$ *such that* $x <//(z, y)$ *and* $//(w, x) < y$, *then* not $x <//(z, y)$ *and* not $//(w, x) < y$.

-

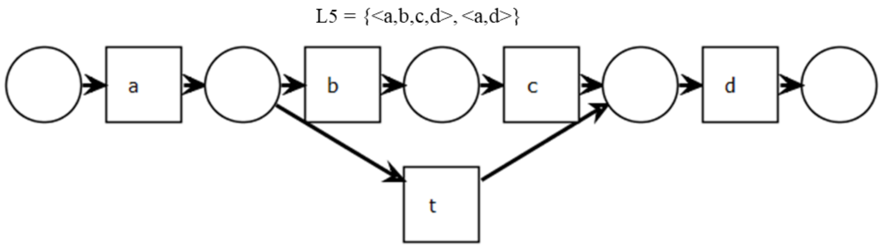


Figure 6.4. Long-skip invisible task in a sequential workflow

Table 6.7. The A/D Table of L5

Task id	$\sigma_1(a,1)$	$\sigma_2(a,1)$	$\sigma_1(b,1)$	$\sigma_1(c,1)$	$\sigma_1(d,1)$	$\sigma_2(d,1)$
T_a	∅	∅	a	b	c	a
T_d	b	d	c	d	∅	∅

Based on Table 6.7 and definition 6.1, we obtain the following dependencies: *a>b, b>c, c>d, a>d, not a<V(b,d)*, and *not V(a,c)<d*. Since we have (*a>b, b>c, c>d*) and (*a>d*), and no concurrency or choice is detected, there exists a long-skip invisible task that skips the two tasks *b* and *c* between task *a* and task *d*. Accordingly, the rule that discovers an *IvT-LS* in a sequential workflow is constructed as follows:

$$\textit{IF there are } w,x,y,z \in \sigma_k \textit{ such that } w > x \textit{ AND } x > y \textit{ AND } y > z \textit{ AND } w > z \textit{ AND } \textit{not } w < V(x,z) \textit{ AND not } V(w,y) < z, \textit{ THEN} \ni t \in IvT - LS_{seq}/x,y \textit{ such that } w < V(x,t) \textit{ AND } V(y,t) < z \text{ (R7)}$$

Rule (7) investigates whether four activities *w*, *x*, *y*, and *z* follow each other successively with causal dependency such that the first activity is followed with causal dependency by the fourth activity, and the four activities are not involved in an XOR-split or XOR-join. If these conditions are fulfilled, then there exists an invisible task *t* that allows the execution of the two in-between activities to be skipped. The detected invisible task will have the following dependencies: an exclusive choice between *x* and *t* (XOR-split) and an exclusive choice between *y* and *t* (XOR-join).

Rule (7) can detect an $IvT - LS_{seq}$ of length two (that is, an invisible task that allows the execution of only two tasks to be skipped), when in fact several tasks can be skipped at once. Actually, this can be drawn easily from rule (7). Before we address this, we introduce a new relation (>>) in definition 6.2, as follows.

Definition 6.2. *Let L be a complete event log and* $\sigma_k \in L$ *be the event traces.*
If there are $a, b, \ldots, y, z \in \sigma_k$ *such that* $a > b$ *and* $b > c$ *and…and* $y > z$, *then* $a >> z$ *and* $(b, c, .., y) \in >>$.

Based on rule (6) and definition 6.2, we define the heuristic rule that can discover an invisible task allowing the execution of several tasks to be skipped as follows:

IF $\exists x_i \in \sigma_k$ *such that* $x_1 >> x_n$ *AND* $x_2, \ldots, x_{n-1} \in >>$ *AND* $x_1 > x_n$ *AND* $x_2, \ldots, x_{n-1} \notin$ *<V(,) AND* $x_2, \ldots, x_{n-1} \notin$ *V(,)<, THEN* $\ni t \in IvT - LS_{seq}$ *such that* $x_1 < V(x_2, t)$ *AND* $V(x_{n-1}, t) < x_n$ (R7*)

Rule (7*) states that, if there exist multiple activities following each other successively with causal dependency, such that the first activity is followed with causal dependency by the last activity, and none of the activities are involved in an XOR-split or XOR-join, then there is an invisible task *t* that allows the execution of all activities except the first and last to be skipped. *t* will be placed such that there is an exclusive choice between *t* and the second activity (XOR-split) and between *t* and the activity before the last activity (XOR-join).

Similar to rule (4), let us assume that the place after *c* and *t* in Fig. 8 is the sink place, and that task *d* does not exist, or the place before *b* and *t* is the source place and that *a* does not exist. Rule (7*) will not be able to detect this case. Therefore, we define rule (8), which is a special case of rule (7*):

IF there are $x_i \in \sigma_k$ *such that* $x_1 >> x_n$ *AND* $x_2, \ldots, x_{n-1} \in >>$ *AND* $[x_1: O$ *AND* $x_n: O]$ *OR* $[I: x_1$ *AND* $I: x_n]$ *THEN* $\ni t \in IvT - LS_{seq}$ *such that* $[x_1 < V(x_2, t)$ *AND* $t: O]$ *OR* $[V(x_{n-1}, t) < x_n$ *AND* $\mathrm{I}: t]$ (R8)

Rule (8) attempts to detect an invisible task of type skip that allows the execution of the sequence $x_2, \ldots, x_n$ to be skipped when x_n is directly connected to the sink place and the execution of the sequence $x_1, \ldots, x_{n-1}$ to be skipped when x_1 is directly connected to the source place.

Let us move now to the case of a concurrent workflow. To discover an *IvT-LS* in a concurrent workflow, we follow the same process used for an *IvT-SS*$_{par}$. Consider the Petri net model in Figure 6.5 and its complete event log L_6. We obtain the following dependencies: *b*>c, *d>e*, *a<//(b, d)*, and *//(c, e)<f*. These dependencies indicate the existence of concurrent tasks (*b>c* and *d>e*), an AND-split in task *a*, and an AND-join in task *f*. If this is so, when task *b*, *c*, *d* or *e* appears in an event trace, all three tasks should appear in succession, e.g., *abcdef*, *adbcef*, *abdecf*. However, a trace such as <a, d, e, f>, where *d* followed by *e* appear alone, might exist in an event log. This can happen when there is an invisible task that allows the execution of *b* and *c* to be skipped. Based on this observation, the heuristic rule to detect an *IvT-LS* in a concurrent workflow is defined as follows:

IF there are $w, x_1, x_2, .., x_n, y_1, y_2, .., y_m, z \in \sigma_k$ *such that* $w <//(x_1, y_1)$ *AND* $//(x_n, y_m) < z$ *AND* $x_1 \gg x_n$ *AND* $y_1 \gg y_m$ *AND* $\exists \sigma_i \subseteq \sigma_k$: $(for\ T_a(y_1)_{\sigma_i} = w{:}\ T_d(y_m)_{\sigma_i} = z)$, *THEN* $\ni t \in IvT - LS_{par} such\ that\ w < V(x_1, t)\ AND\ V(x_n, t) < z$ (R9)

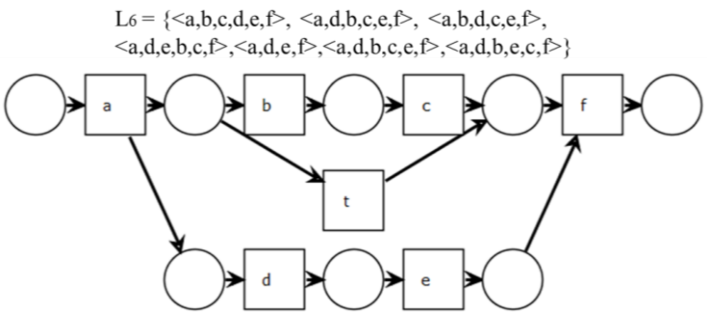

L6 = {<a,b,c,d,e,f>, <a,d,b,c,e,f>, <a,b,d,c,e,f>,
<a,d,e,b,c,f>,<a,d,e,f>,<a,d,b,c,e,f>,<a,d,b,e,c,f>}

Figure 6.5. Long-skip invisible task in a concurrent workflow

Rule (9) states that, if activities $x_1, x_2, x_3, .., x_n$ are in parallel with $y_1, y_2, y_3, .., y_m$, such that $x_1 > x_2$, $x_2 > x_3$,.., $x_{n-1} > x_n$ and $y_1 > y_2$, $y_2 > y_3$,.., $y_{m-1} > y_m$; if there is an event trace σ_i in which the ancestor of y_1 is the activity splitting the AND of concurrency; and the descendent of y_m is the activity joining the AND of concurrency, then there is an invisible task *t* that enables the execution of activities $x_1, x_2, x_2, .., x_n$ to be skipped, such that there is an exclusive choice between x_1 and *t* (XOR-split) and between x_n and *t* (XOR-join). Rule (9) can detect an invisible task of type long skip that allows multiples activities in parallel to be skipped.

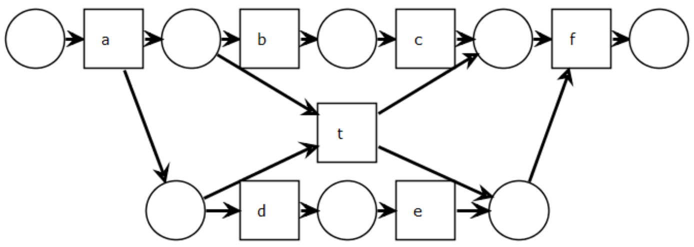


Figure 6.6. Process model with an invisible task of type long skip combined with a parallel construct

Similar to *IvT-SSpar*, there is also another type of *IvT-LSpar* where *t* is combined with a parallel construct as shown in Figure 6.6. The rule that can detect this type of invisible tasks is as follows.

IF there are $w, x_1, x_2, .., x_n, y_1, y_2, .., y_m, z \in \sigma_k$ *such that* $w <//(x_1, y_1)$ *AND* $//(x_n, y_m) < z$ *AND* $x_1 \gg x_n$ *AND* $y_1 \gg y_m$ *AND* $w > z$ *THEN* $\ni t \in IvT - LS_{par}$ *such that* $w < V(x_1 \wedge y_1, t)$ *AND* $V(x_n \wedge y_m, t) < z$ (R10)

As can be seen, the rules that can detect invisible tasks of type long skip are based on the relation (>). However, if the execution of two tasks can be skipped by the *IvT-LS* and the execution of the same tasks can be repeated by the existence of an *IvT-LR*, then the invisible task of type long skip cannot be detected. To detect *IvT-LR*, the relation (>) cannot be used because the two tasks are following each other. Hence, the *IvT-LS* cannot be detected. We call this situation an *overlapping*. This overlapping can be handled by the introduction of the following rule:

IF $\exists t \in IvT - LR/x, y$ AND $w > x$ AND $T_d(x) = y$ AND $y > z$ AND $w > z,$ THEN $\exists t \in IvT - LS/x, y$ (R11)

This rule enables us to detect an *IvT-LS* that allows the execution of two tasks to be skipped when the execution of these tasks can be repeated with the existence of an *IvT-LR*.

(e) Detection of Invisible Tasks of Type Side (IvT-SD).

There are situations in which the workflow begins with two or more tasks that must be executed simultaneously or ends with concurrent tasks. In these situations, the representation of the workflow net requires an invisible task that is placed before or after the concurrent tasks. Therefore, *an IvT-SD* can belong to one of two categories: an invisible task of type side in the beginning (*IvT-SDB*) that directly follows the source place or an invisible task of type side at the end (*IvT-SDE*) that is followed directly by the sink place.

To detect an *IvT-SDB*, consider the complete log L7 shown in Figure 6.7(a). The A/D Table of log L7 is depicted in Table 6.8. The relations extracted from the A/D Table are $I{:}\,a$, $I{:}\,b$ and *//(a,b)<c*, indicating that there is an AND-join in task *c* of two parallel tasks *a* and *b*, while no AND-split can be identified from the A/D Table of L7. However, the parallel tasks *a* and *b* are connected directly to the source place without an AND-split. This is possible only when there exists an invisible task that splits the AND-split of the two parallel tasks *a* and *b*. On this basis, the rule that discovers an invisible task of type side in the beginning is defined as follows:

IF there are $x_1, x_2, .., x_n, y_1, y_2, .., y_m, z \in \sigma_k$ *such that* $x_1 \gg x_n$ *AND* $y_1 \gg y_m$ *AND* $//(x_n, y_m) < z$ $I{:}\,x_1$ *AND* $I{:}\,y_1$*, THEN there exists* $t \in IvT - SDB$ *such that* $t <//(x_1, y_1)$

(R12)

Table 6.8. The A/D Table of L7

Task id	$\sigma_1(a,1)$	$\sigma_2(a,1)$	$\sigma_1(b,1)$	$\sigma_2(b,1)$	$\sigma_1(c,1)$	$\sigma_2(c,1)$
T_a	Ø	b	a	Ø	b	a
T_d	b	c	c	a	Ø	Ø

Rule (12) investigates that, if activities $x_1, x_2, x_3, .., x_n$ are in parallel with $y_1, y_2, y_3, .., y_m$, such that $x_1 > x_2$, $x_2 > x_3$,.., $x_{n-1} > x_n$ and $y_1 > y_2$, $y_2 > y_3$,.., $y_{m-1} > y_m$; if there is an AND-join of x_n and y_m and no AND-split of x_1 and y_1 (x_1 and y_1 do not have an ancestor); then there exists an invisible task that is connected directly to the source place and is followed directly by the concurrent tasks x_1 and y_1.

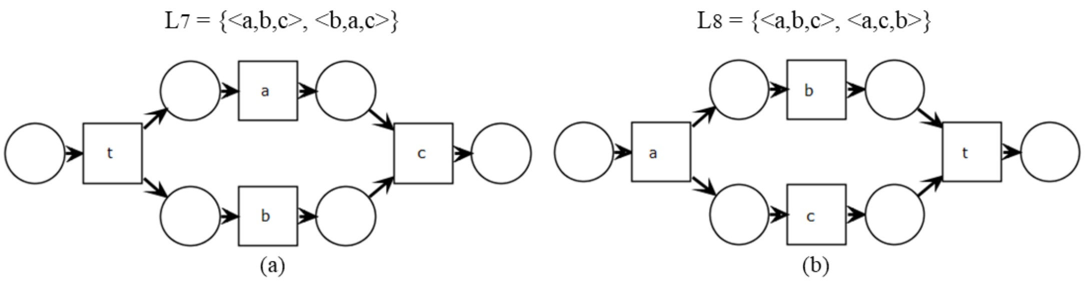


Figure 6.7. Sound process models with an IvT-SDB (a) and an IvT-SDE (b)

To identify an *IvT-SDE*, consider the complete event log L8 and its corresponding model shown in Figure 6.7(b). The A/D Table of L8 is presented in Table 6.9. The relations derived from Table 6.9 are a>//(b,c), b:O, and c:O. Unlike in an *IvT-SDB*, there is an AND-split in task *a* of two concurrent tasks *b* and *c*, whereas no AND-join can be derived from the A/D Table. However, the parallel tasks *b* and *c* are connected directly to the sink place, which can be allowed only if there exists an invisible task that joins them. Accordingly, the heuristic rule that can discover such a construct is designed as follows:

IF there are $x_1, x_2, .., x_n, y_1, y_2, .., y_m, z \in \sigma_k$ *such that* $z <//(x_1, y_1)$ *AND* $x_1 \gg x_n$ *AND* $y_1 \gg y_m$ *AND* $x_n: O$ *AND* $y_m: O$, *THEN there exists* $t \in IvT - SDE$ *such that* $//(x_n, y_m) < t$

(R13)

Rule (13) verifies that, if activities $x_1, x_2, x_3, .., x_n$ are in parallel with $y_1, y_2, y_3, .., y_m$, such that $x_1 > x_2$, $x_2 > x_3$,.., $x_{n-1} > x_n$ and $y_1 > y_2$, $y_2 > y_3$,.., $y_{m-1} > y_m$; if there is an AND-split of x_1 and y_1 and no AND-join of x_n and y_m (x_n and y_m do not have a descendent); then there exists an invisible task joining the concurrent tasks x_n and y_m and connected directly to the sink place.

Table 6.9. The A/D Table of L8

Task id	$\sigma_1(a,1)$	$\sigma_2(a,1)$	$\sigma_1(b,1)$	$\sigma_2(b,1)$	$\sigma_1(c,1)$	$\sigma_2(c,1)$
T_a	∅	∅	a	c	b	a
T_d	b	c	c	∅	∅	b

(f) Detection of Invisible Tasks of Type Switch (IvT-SW).

An invisible task of type switch is a hidden task that allows a workflow net to switch the execution of some tasks to others. We consider the complete log L9 and its model depicted in Figure 6.8. The invisible task indicated in the process model switches the execution from task *d* to *c*. The A/D Table of L9 is shown in Table 6.10. The relations that can be extracted from this table are *a>c, b>d, b>c, //(a,b)<c,* and *b<//(d,c)*. As can be seen, the derived relations refer to the existence of an AND-split in task *b* of two parallel tasks *d* and *c* and an AND-join in task *c* of two parallel tasks *a* and *b*, which is unreasonable. Based on definition 6.1, we obtain *not//(a,b)<c* and *not b<//(d,c)*.

Accordingly, we have the following dependencies: *a>c, b>d,* b>c, *not //(a,b)<c,* and

not *b<//(d,c)*. Since no parallel or choice is determined, there exists an IvT that switches the execution from task d to task c. Based on this, the rule that can derive an *IvT-SW* is constructed as follows:

$$IF\ there\ are\ w, x, y, z \in \sigma_k\ such\ that\ w > y\ AND\ x > z\ AND\ x > y\ AND\ not\ x <//(z, y)\ AND\ not//(w, x) < y,\ THEN\ there\ exists\ t \in IvT - SW\ such\ that\ x < V(t, z)\ AND\ V(w, t) < y \quad \text{(R14)}$$

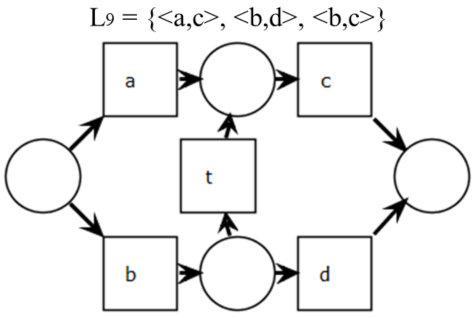


Figure 6.8. Example of a sound process model containing an IvT-SW

Table 6.10. The A/D Table of L_9

Task id	$\sigma_1(a,1)$	$\sigma_1(c,1)$	$\sigma_2(b,1)$	$\sigma_3(b,1)$	$\sigma_2(d,1)$
T_a	∅	a	∅	∅	b
T_d	c	∅	d	c	∅

Rule (14) verifies that, if there exist tasks *w, x, y,* and *z* such that the causal dependencies between them are $w > y$, $x > z$, *and* $x > y$ and such that they are not involved in any parallel or choice, then an invisible task of type switch will be detected such that *x* is splitting the exclusive choice between *t* and *z*, and *y* is joining the exclusive choice between *w* and *t*.

6.3.2. Short Loops Discovery

A short loop can be a loop of length one, L_{1p} (when one task can be repeated several times), or a loop of length two, L_{2p} (when two tasks can be repeated many times). These two types of short loops will be detected by our heuristic rules below.

(a) Detection of a Loop of Length One (L_{1p}).

To detect a loop of length one L_{1p}, consider the complete log L10 and its process model shown in Figure 6.9, in which task b can be repeated several times. The A/D Table of L10 is presented in Table 6.11 and allows us to obtain $a>c$, $a>b$, $b>c$, and $T_d(b) = b$. Although we have $a>b$ and $a>c$, the conditions of AND-split and XOR-split are not satisfied. Similarly, for $a>c$ and $b>c$, the conditions of AND-join and XOR-join are not satisfied. However, the relation $T_d(x) = x$ can also refer to the existence of an invisible task of type short redo. The only difference is that a succession such as *ac* can exist in an event log with L_{1p} but not with an invisible task of type short redo. To distinguish between L_{1p} and an *IvT-SR*, it is necessary to investigate the existence of a succession such as *ac*. Based on this, letting L be a complete log, we define the heuristic rule that can discover loops L_{1p} of length one as follows:

IF there are $x, y, z \in \sigma_k$ *such that* $x > z$ *AND* $x > y$ *AND* $y > z$ *AND* not $x <//(y,z)$ AND not $x < V(y,z)$ AND not $//(x,y) < z$ AND not $V(x,y) < z$ *AND there exist* $\sigma_i \subseteq \sigma_k$ *such that* $T_d(y)_{\sigma_i} = y$ *OR* $T_a(y)_{\sigma_i} = y$*, THEN* $y \in L_{1p}$ (R15)

Rule (15) states that, if there exist three tasks *x, y,* and *z* such that there are causal dependencies between *x* and *y*, between *x* and *z*, and between *y* and z; the three activities are not involved in an AND-split, AND-join, XOR-split, or XOR-join; and the descendant or ancestor of *y* is equivalent to *y*, then there exists a loop L_{1p} in *y.*

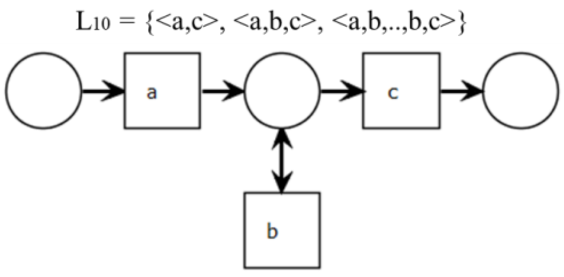


Figure 6.9. Sound workflow models containing a short loop of length one $\boldsymbol{L_{1p}}$

Table 6.11. The A/D Table of L10

Task id	$\sigma_1(a,1)$	$\sigma_2(a,1)$	$\sigma_3(a,1)$	$\sigma_1(b,1)$	$\sigma_2(b,2)$	$\sigma_2(b,k)$	$\sigma_1(c,1)$	$\sigma_2(c,1)$	$\sigma_3(c,1)$
$\boldsymbol{T_a}$	∅	∅	∅	a	a	b	b	b	a
$\boldsymbol{T_d}$	b	b	c	c	b	c	∅	∅	∅

However, if all conditions of rule (15) are satisfied except the last two ($T_d(y)_{\sigma_i} = y$ or $T_a(y)_{\sigma_i} = y$), then an invisible task of type short skip will be detected. On the other hand, if $T_d(y)_{\sigma_i} = y$ or $T_a(y)_{\sigma_i} = y$ is satisfied, an invisible task of type short redo will be detected. Thus, if all conditions of rule (15) are satisfied, two invisible tasks will be detected: one that allows the execution of a given task to be repeated, and one that allows the execution of that task to be skipped. Hence, one can conclude that if, rule (15) is satisfied, L_{1p} can never be discovered. This case also refers to the overlapping.

Examine Figure 6.10. The event log depicted is similar to $L_{10,}$ (i.e., the event log of L_{1p}). The models illustrated in Figure 6.9, and Figure 6.10 share similar behaviour but differ in structure. One of the main quality criteria used for process discovery is simplicity (Van der Aalst, 2011). A good model has good fitness, good precision, good generalization, and a simple structure. Among the two models in Figure 6.9, and Figure 6.10, the most structurally simple is the model of L_{1p} in Figure 6.9. Hence, we introduce the following new rule:

$$IF\ t\ \in IvT - SS/y\ AND\ t\ \in IvT - SR/y,\ THEN\ y \in L_{1p} \quad \text{(R16)}$$

Rule (16) states that, if an invisible task of type short skip (which allows the execution of task y to be skipped) and an invisible task of type short redo (which allows the execution of task y to be repeated) are detected, then they will be transformed into a loop of length one L_{1p} in y.

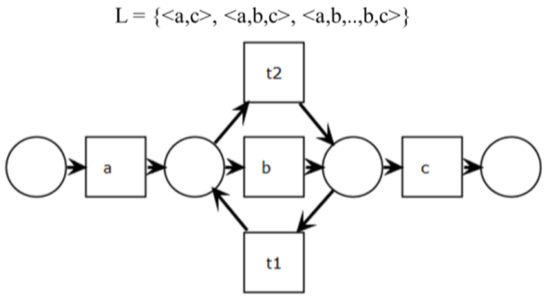


Figure 6.10. Process model having similar event logs and behaviour with $\boldsymbol{L_{1p}}$

(b) Detection of a Loop of Length Two (L_{2p}).

First, we divide loops of length two (*L_{2p})* into two types, *$L_{2p,a}$* and *$L_{2p,b}$*, as shown in Figure 6.11. Loops *$L_{2p,a}$* and *$L_{2p,b}$* have the same definition and the same role; however, they are represented differently. Therefore, they will be handled separately.

To detect a loop of length one of type $L_{2p,a}$, consider the complete log L_{11} and its workflow model depicted in Figure 6.11 (a), where tasks b and c are in a loop and can be repeated many times. The A/D Table of log L_{11} is presented in Table 6.12. Accordingly, we obtain only $a>b$ and $b>d$. Since $T_d(c,1)_{\sigma_2} = b$ and $T_d(b,1)_{\sigma_2} = c$, the relation (>) cannot be used to gain insight into tasks b and c. Therefore, the heuristic rule that can detect loop of length two type $L_{2p,a}$ will be strongly based on direct succession relations. As can be seen, $T_d(c,1)_{\sigma_2} = b$ and $T_d(b,1)_{\sigma_2} = c$ represent the ordering bcb. However, this ordering can also exist in the case of an invisible task of type long redo $bcbc$. The difference between two tasks b and c belonging to an *IvT-LR* and two tasks b and c belonging to $L_{2p,a}$ is that the descendent of the last occurrence of b in an *IvT-LR* is task c, whereas the descendent of the last occurrence of b in L_{2p} is different than task c. Based on this, let k be the kth occurrence of b and c, let L be an event log, and let $\sigma_i \in L$ be the event traces. Based on these observations, the heuristic rule (17) to discover a loop of length two is defined as follows.

IF $\exists x, y \in \sigma_i$ *and* $\exists k \in \{1, .., p\}$ *such that* $T_d(x,k)_{\sigma_i} = y$ *AND* $T_d(y,k)_{\sigma_i} = x$ *AND* $T_d(x,k+1)_{\sigma_i} \neq y$ *AND* $T_a(x,1)_{\sigma_i} \neq y$, *THEN there exist* $x, y \in L_{2p,a}$ (R17)

Rule (17) verifies four conditions to detect the existence of a loop of length two of type $L_{2p,a}$. The first and second conditions verify that the descendent of the kth occurrence of x is y and the descendent of the kth occurrence of y is x, respectively. The third condition verifies that the descendent of the $(k+1)$th occurrence of x is different than y, and the last condition verifies that the ancestor of the first occurrence of x is distinct from y. The four conditions are investigated in the same event trace σ_i. If a $k \in \{1, .., p\}$ is found in which the four conditions are satisfied, the existence of a loop of length two between x and y will be detected. For instance, let L = {<a,b,d>, <a,b,c,b,c,b,c,b,d>} be a sample event log of a model containing $L_{2p,a}$. The value of k that satisfies the conditions of rule (17) is k=3 in the second event trace.

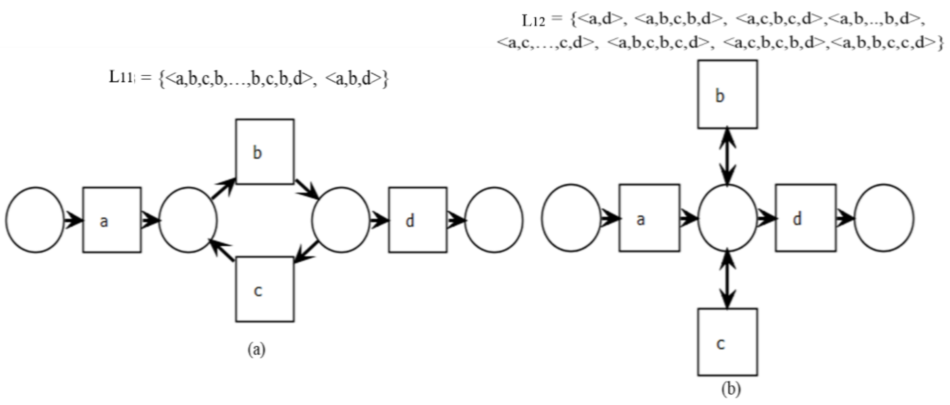

L12 = {<a,d>, <a,b,c,b,d>, <a,c,b,c,d>,<a,b,..,b,d>, <a,c,…,c,d>, <a,b,c,b,c,d>, <a,c,b,c,b,d>,<a,b,b,c,c,d>}
L11 = {<a,b,c,b,…,b,c,b,d>, <a,b,d>}
a
b
c
d
(a)
a
b
c
d
(b)

Figure 6.11.The workflow model of a loop of length two **:** (a) $\boldsymbol{L_{2p,a}}$ and (b) $\boldsymbol{L_{2p,b}}$

Table 6.12. The A/D Table of L11

T id	$\sigma_1(a,1)$	$\sigma_2(a,1)$	$\sigma_1(b,1)$	$\sigma_2(b,1)$	$\sigma_2(b,2)$	$\sigma_2(b,k)$	$\sigma_2(c,1)$	$\sigma_2(c,k)$	$\sigma_1(d,1)$	$\sigma_2(d,1)$
$\boldsymbol{T_a}$	∅	∅	a	a	c	c	b	b	b	b
$\boldsymbol{T_d}$	b	b	d	c	c	d	b	b	∅	∅

The second type of loop length two $L_{2p,b}$ as depicted in Figure 6.11(b) can be seen as a combination of two loops of length one. The event log of a process model containing this kind of loop can have the following behaviours: *bcbc*, *cbcb*, *cbc*, *bcb*, etc. These behaviours can be found also when either a loop of length two of type $L_{2p,a}$ or invisible tasks of type long redo *IvT-LR* exists in the event log (an overlapping situation). Based on this, we define the following rule for detecting a loop length two of type $L_{2p,b}$:

$$IF\ x,y \in (L_{2p,a}\ OR\ IvT - LR)\ AND\ x\ \in\ L_{1p}\ AND/OR\ y\ \in\ L_{1p},\ THEN\ x,y \in L_{2p,b}\ \text{(R18)}$$

Rule (18) states that if with the heuristic rules *x* and *y* are detected to belong to loop of length two of type $L_{2p,a}$ or to invisible task of type long redo; and *x* and/or *y* are (is) detected to belong to loop of length one, then *x* and *y* belong to loop of length two of type $L_{2p,b}$ not to $L_{2p,a}$ nor $IvT - LR$. This indicates that this type of loop can be detected only after invisible tasks of type long redo and loop length two of type $L_{2p,a}$ are detected.

6.3.3. Non-Free Choice Construct (NFC) Discovery

A non-free choice construct is a case in which choice and synchronization are combined (Wen *et al,* 2007a). As mentioned before, the detection of implicit dependencies is necessary to discover a workflow net with non-free choice constructs. Let us assume that there is an implicit dependency between activity *A* and activity *B.* If task *A* is executed, other tasks should be executed before *B* is executed. Since the distance of the dependency between *A* and *B* is at least two, there will be no trace in the event log where *A* is directly followed by *B.*

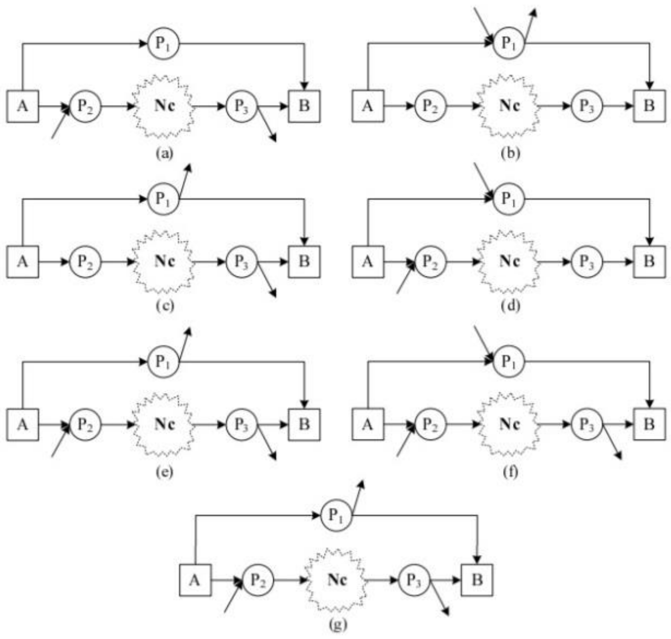


Figure 6.12. Sound sub-workflow nets with implicit dependencies (Wen et al., 2006)

Wen et al., (2006) classified implicit dependencies into seven cases, as shown in Figure 6.12, such that each pattern appears in a sound workflow net. Since the detection of implicit dependencies is considered the most important factor in appropriately discovering process models with non-free choice constructs, the seven cases of implicit dependencies depicted in Figure 6.12 will be detected with our heuristic rules. Before that we denote $T_{d+k}(\mathrm{a})_{\sigma_j}$ and $T_{a+k}(\mathrm{a})_{\sigma_j}$ the (k-1)th descendent and ancestor of task a in the trace σ_j respectively (e.g. $T_{d+1}(\mathrm{a})_{\sigma_j}$denotes the second descendent of a in trace σ_j). The heuristic rules that allow detecting implicit dependencies are as follows:

$$IF\ \exists w,x,y,z \in\ \sigma_k\ such\ that\ w <//(x,y)\ AND\ \exists\ \sigma_i \subseteq\ \sigma_k{:}\ T_a(x)_{\sigma_i} = w\ AND\ \exists\ k{:}$$
$$T_{d+k}(x)_{\sigma_i} = z\ AND\ \nexists k'{:}\ T_{d+k'}(x)_{\sigma_i} = y\ THEN\ w > z\ is\ an\ implicit\ dependency \quad \text{(R19)}$$

$$IF\ \exists w,x,y,z \in\ \sigma_k\ such\ that\ //(x,y) < w\ \ AND\ \exists\ \sigma_i \subseteq\ \sigma_k{:}\ T_d(x)_{\sigma_i} = w\ AND\ \exists\ k{:}$$

$T_{a+k}(x)_{\sigma_i} = z$ *AND* $\nexists k': T_{a+k'}(x)_{\sigma_i} = y$ *THEN* $w > z$ *is an implicit dependency* (R20)

IF $\exists v, w, x, a, y, z \in \sigma_k$ $V(v, w) < x$ *AND* $x >>> a$ *AND* $a < V(y, z)$ *AND* [$\nexists$ $\sigma_i \subseteq \sigma_k$: *for* $T_a(x)_{\sigma_i} = v$: $T_d(a)_{\sigma_i} = z$], *THEN* $v > y$ *is an implicit dependency* (R21)

Rule (19) detects implicit dependencies from an event log of a workflow model with a sub-workflow net similar to the cases shown in Figures 6.3.12 16(b), (c), and (g). This rule investigates whether x and y are in parallel and the AND-split is in w, whether there is a trace σ_i in which the ancestor of x is w, whether there exists a k such that (k-1)th descendent of x in σ_i is z, and whether there is no k' such that (k'-1)th descendent of x in σ_i is y. If all of these conditions are satisfied, an implicit dependency w>z will be detected.

Rule (20) detects implicit dependencies from an event log of a process model with a sub-workflow net similar to the cases shown in Figure 6.12(d). The explication of this rule is oppositely similar to the explanation of rule (19).

Finally, rule (21) detects implicit dependencies from an event log of a workflow model with a sub-workflow net similar to Figures 6.3.12(a), (e) and (f). This rule states that, if there exists an exclusive choice between two tasks v and w joined in x, and x>>a, and a splits the exclusive choice of two tasks y and z, and there exists no trace σ_i in which the ancestor of x is v and the descendent of a is z, then an implicit dependency $v > y$ will be detected. Note that a in rule (21) can be identical to x.

The three rules (19), (20) and (21) are developed to detect the implicit dependencies of the seven types shown in Fig.15. However, these dependencies need to be detected after the invisible tasks are detected. This way, non-free choice constructs can be discovered correctly.

Let us take the examples shown in Figure 6.13 to illustrate the capabilities of these rules. The workflow nets illustrated in Figures 6.13(a) and (d) do not have any implicit dependencies, while those shown in Figures 6.13(b), (c), (e), (f), and (g) contain implicit dependencies. Figure 6.13(a) depicts an original workflow net. Its corresponding complete event log is {<A,C,F,D>, <A,F,C,D>, <B,C,G,E>, <B,G,C,E>}. When we apply the three heuristic rules, no implicit dependency is detected. Similarly, for Figure 6.13(d) and its corresponding event log {<A,C,B,D>, <A,B,C,D>, <A,E,D>}, no implicit dependency is identified. For Figure 6.13(b), the corresponding complete workflow log is {<A,B,C>, <A,D,B,E>, <A,B,D,E>}. From this log, the implicit dependency $A > C$ is detected with heuristic rule (R19). For Figure 6.13(c), the corresponding complete event log is {<A,C,D>, <B,C,E>, <A,C,F,E>, <A,F,C,E>}. From this log, two implicit dependencies $A > D$ and $B > E$ are detected with rule (R21). The corresponding complete event log of Figure 6.13(e) is {<A,C,E,B,C,D>}. From this log, the implicit dependencies $A > E$ and $B > D$ are detected with rule (R21). For Figure 6.13(f), for which one complete event log is {<A,E,B,C,D,F>, <A,B,E,C,D,F>, <A,B,C,D,E,F>, <A,B,C,E,D,F>, <A,B,G,D,F>}, the implicit dependency $A > G$ is detected with rule (R19), and the implicit dependency $G > F$ is detected with rule (R20). Finally, the corresponding complete event log of Figure 6.13(g) is {<A,B,C,D,E >, <A,C,D>}. From this log, the implicit dependencies $t_1 > t_2$ and $B > E$ are detected with rule (R21) after the two invisible tasks t_1 and t_2 of type short skip are detected.

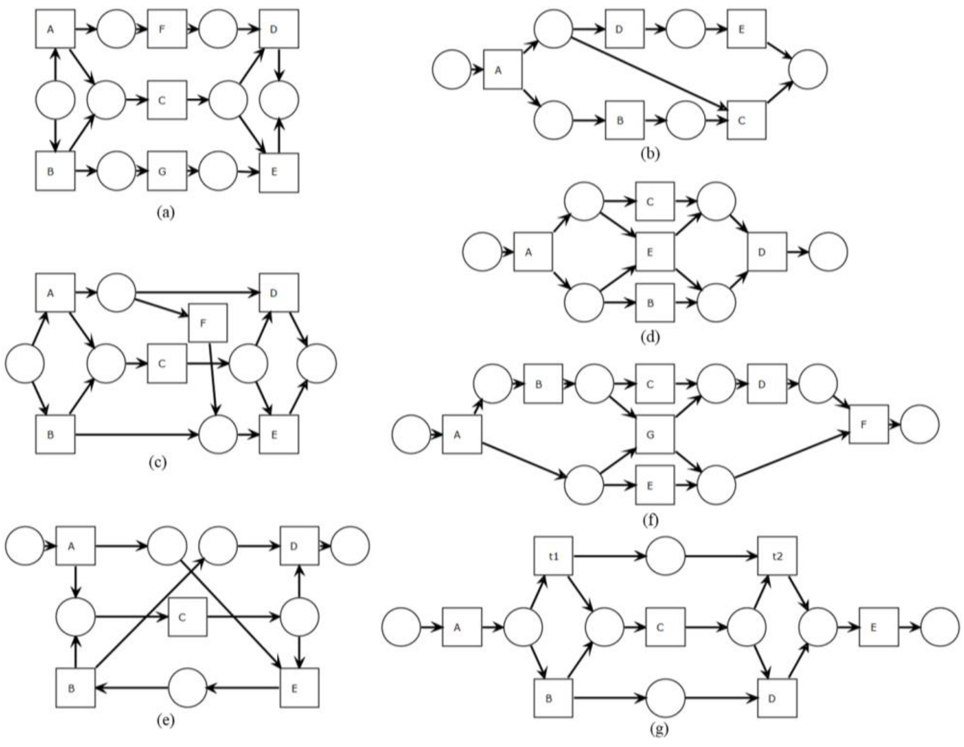


Figure 6.13. Detection of implicit dependencies through the heuristic rules

There is a case where the model discovered with invisible tasks involved in non-free choice using the aforementioned rules is unsound. An example of this case is illustrated in Figure 6.14.

Figure 6.14(a) depicts the original sound model, while Figure 6.14(b) shows the process model discovered with the heuristic rules. In the model of Figure 6.14(b), invisible tasks t_1 and t_2 are detected first. Then the implicit dependencies $A > D$ and $B > E$ of the non-free choice construct are detected. These implicit dependencies make the model unsound. The developed heuristic rules does not detect the arcs connecting p3 to t_1 and p4 to t_2. To solve this problem and discover a sound model, the following rule is developed:

IF $\exists v, w, x, a, y, z \in \sigma_k$: $V(v,w) < x$ *AND* $x>>a$ *AND* $a< V(y,z)$ *AND* $v>y$ *AND* $\exists t \in IvT$: $a< V(t,y)$, *THEN* $v > t$ *is an implicit dependency* (R22)

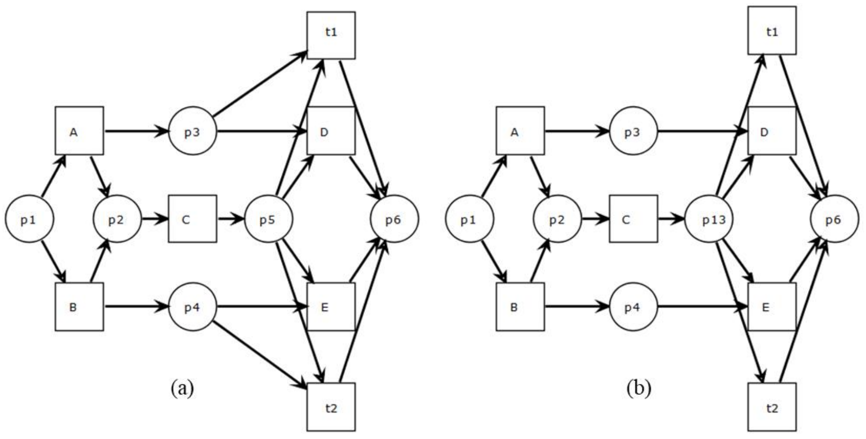


Figure 6.14. Process model with invisible tasks involved in a non-free choice construct: (a) original model and (b) discovered model

6.3.4. Duplicate Tasks (DT) Discovery

Duplicate tasks are those sharing the same name but placed in different transitions in the workflow net. Figure 6.15 depicts a process model with two duplicate tasks, *a* and *e*. It is difficult to differentiate between duplicate tasks sharing the same name in the event log. Hence, it is difficult to correctly mine the process model in Figure 6.15. Fortunately, Li *et al*., (2007) handled this problem by first using heuristic rules to identify tasks with similar names and then renaming them differently in the log. They then used the $\alpha -$algorithm to mine a model from the updated log. The authors developed three rules to detect duplicate tasks, which we will transform here into our notations; to see the original notations, please refer to (Li *et al*., 2007). Let L be a complete event log and $\sigma_k \in L$ be the event traces. Let $\sigma_k(x,n)$ be the '*nth*' occurrence of task *x* in the trace σ_k. The heuristic rules are as follows:

IF $\exists x \in \sigma_i, x \in \sigma_j$ *such that* $T_a(x,n_1)_{\sigma_i} \neq T_a(x,n_2)_{\sigma_j}$ *AND* $T_d(x,n_1)_{\sigma_i} \neq T_d(x,n_2)_{\sigma_j}$ *AND* $T_a(x,n_1)_{\sigma_i} \neq T_d(x,n_2)_{\sigma_j}$ *AND* $T_a(x,n_2)_{\sigma_j} \neq T_d(x,n_1)_{\sigma_i}$, *THEN* $< (x,n_1)_{\sigma_i},\ (x,n_2)_{\sigma_j} > \in DT$

(R23)

$$IF\ \exists x \in \sigma_i, x \in \sigma_i \text{ and } i \neq j \text{ such that } T_a(x,n_1)_{\sigma_i} = T_d(x,n_2)_{\sigma_j} \text{ AND } [T_a(x,n_2)_{\sigma_j} \neq T_{a+1}(x,n_1)_{\sigma_i} \text{ OR } T_d(x,n_1)_{\sigma_i} \neq T_{d+1}(x,n_2)_{\sigma_j}], THEN < (x,n_1)_{\sigma_i}, (x,n_2)_{\sigma_j} >\in DT \quad \text{(R24)}$$

$$IF\ \exists x \in \sigma_i, x \in \sigma_i \text{ and } i \neq j \text{ such that } T_d(x,n_1)_{\sigma_i} = T_a(x,n_2)_{\sigma_j} \text{ AND } [T_a(x,n_1)_{\sigma_i} \neq T_{a+1}(x,n_2)_{\sigma_j} \text{ OR } T_d(x,n_2)_{\sigma_j} \neq T_{d+1}(x,n_1)_{\sigma_i}], THEN < (x,n_1)_{\sigma_i}, (x,n_2)_{\sigma_j} >\in DT \quad \text{(R25)}$$

Rule (23) states that, if the ancestors and descendants of two tasks sharing the same name (x) are distinct and *are not cross-equivalent* (i.e., $T_a(x)_{\sigma_i} \neq T_d(x)_{\sigma_j}$), then x with occurrence n$_1$ and x with occurrence n$_2$ are duplicate tasks sharing the same name. *Cross-equivalence* refers to the case in which the ancestor of (x,n$_1$) in σ_i is equivalent to the descendent of (x,n$_2$) in $\sigma_{j,}$ and vice versa.

In Rule (24), the first condition investigates the case in which the ancestor of (x,n$_1$) in σ$_i$ and the descendant of (x,n$_2$) in σ$_j$ *are cross-equivalent*. The second condition identifies the difference between the ancestor of (x,n$_2$) in σ_j and the second ancestor of (x,n$_1$) in σ_i. The third condition is similar to the second and states the requirement that the descendant of (x,n$_1$) in σ_i and the second descendant of (x,n$_2$) in σ_j are not equivalent.

Rule (25) handles another similar situation. Li *et al.*, (2007) stated that, if the conditions of the aforementioned rules are satisfied, then task *x* with occurrence n$_1$ and task *x* with occurrence n$_2$ belong to the group of duplicate tasks DT. However, a limitation of the first rule has been discovered. Consider the complete log shown in Figure 6.16. If we apply rule (23) to this log, we find that task *c* belongs to duplicate tasks, when in fact task *c* is a unique task. Although the ancestors and descendants of two occurrences of task *c* are different and have no cross-equivalence, task c is still a unique task. We can conclude that, when a task joins an exclusive choice of distinct tasks and splits an exclusive choice of other different tasks, rule (23) incorrectly returns a duplicate task. Based on these observations, the conditions of rule (23) are insufficient and need to be extended. Therefore, we extend rule (23) to rule (23*) as follows:

$$If\ \exists v,w,x,y,z \in \sigma_k \text{ such that rule (20) is satisfied AND not } V(w,v) < x \text{ AND not } x < V(y,z), THEN < (x,n_1)_{\sigma_i}, (x,n_2)_{\sigma_j} >\in DT \quad \text{(R23*)}$$

We extended rule (23) to rule (23*) by adding the requirement that task *x* does not join an exclusive choice of distinct tasks and also does not split an exclusive choice of other different tasks while keeping rules (24) and (25) the same. If a duplicate task is detected by rules (23*), (24), and (25), it will be renamed in the log. Then, the new log will be mined by our heuristic rule-based algorithm. Note that duplicate tasks must be detected first to not affect the detection of the other constructs.

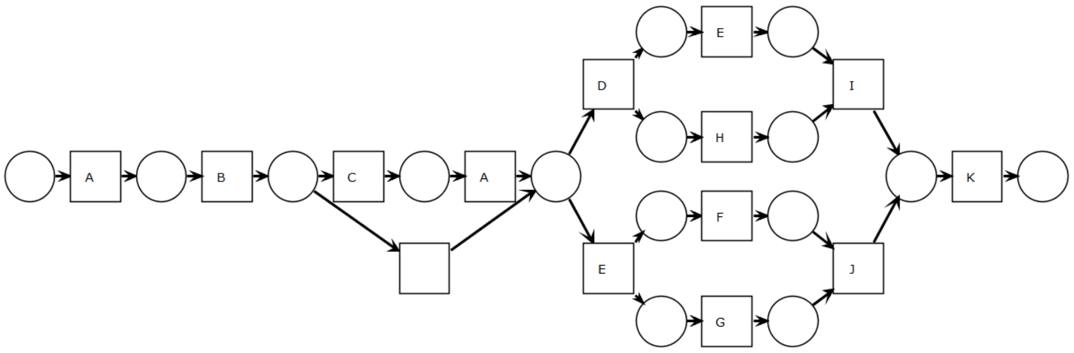


Figure 6.15. Example of a process model with duplicate tasks

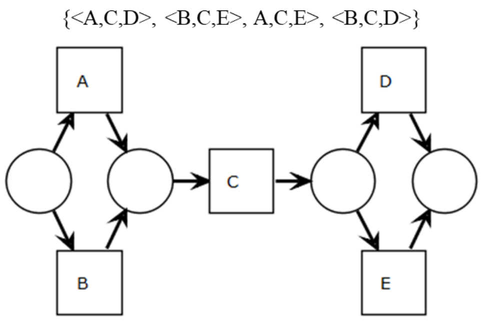


Figure 6.16. A process model with an OR-join and OR-split

6.4. Mining Algorithm

In this section we will present the heuristic rule-based algorithm. Let L be a complete event log and σ be a sequence of activities in L. The algorithm HR is as follows:

Algorithm $\boldsymbol{HR(W.N)}$

```
1.  $T_{log} \leftarrow \{t \in T | \exists_{\sigma \in L} t \in \sigma\}$.
2.  $L^{-DT} \leftarrow L$.
3.  $isDup \leftarrow$ false.
4.  $isIdentify \leftarrow$ false.
5.  $isStandardConstructs \leftarrow$ false.
6.  $isComplexConstructs \leftarrow$ false.
7.  $isOverlap \leftarrow$ false.
8.  FOR $\forall\, t \in T_{log}$ DO
     ($\gamma \leftarrow constructADTable(t, L^{-DT})$.
     FOR $\forall\, \theta \in \gamma$ DO
        ($\gamma' \leftarrow \{\theta' \in \gamma / \theta' \neq \theta\}$.
         FOR $\forall\, \theta' \in \gamma'$ DO
           ($isDup \leftarrow judgeDuplicate(\theta, \theta')$.
            IF $isDup$ THEN
        (//$t'$ is the renamed task of task t
                 $t' \leftarrow renameTask(t, \theta', \gamma)$.
             $T_{log} \leftarrow T_{log} \cup \{t'\}$.
        $isIdentify \leftarrow true.).).)$.
         IF $isIdentify$ THEN
         $L^{-DT} \leftarrow IdentifyLog(L^{-DT}, \gamma).)$.
9.  FOR $\forall\, \theta \in T_{log}$ DO
    ($\gamma' \leftarrow constructADTable(t, L^{-DT})$.).
10. FOR $\forall\, \beta_k \in \gamma'$ DO
        ($isStandardConstructs \leftarrow judgeSTDConst(\beta_k)$.).
11. $T_{first} \leftarrow \{t \in isStandardConstructs / \exists_{\sigma \in L^{-DT}} I : t\}$.
12. $T_{last} \leftarrow \{t \in isStandardConstructs / \exists_{\sigma \in L^{-DT}} t : O\}$.
13. FOR $\forall\, \beta'_k \in \gamma' \wedge isStandardConstructs$ DO
        ($isComplexConstructs \leftarrow judgeCOMPLEXConst(\beta'_k)$.).
14. $D_C \leftarrow \{isStandardConstructs\} \cup \{isComplexConstructs\}$.
15. FOR $\forall\, C \in D_C$ DO
```

```
    $(isOverlap \leftarrow judgeOverlapping(C).)$.
16. $D_C \leftarrow D_C - isOverlap.$
17. $X \leftarrow \{(A,B) \in D_C / \forall (A',B') \in D_C : A \subseteq A' \wedge B \subseteq B' \Rightarrow (A,B) = (A',B')\}$.
18. $P \leftarrow \{P_{(A,B)} / (A,B) \in X\} \cup \{p_{source}, p_{sink}\}$.
19. $F \leftarrow \{(a, P_{(A,B)}) / (A,B) \in X \wedge a \in A\} \cup \{(P_{(A,B)}, b) / (A,B) \in X \wedge b \in B\} \cup$
    $\{(p_{source}, t) / t \in T_{first}\}\} \cup \{(t, p_{sink}) / t \in T_{last}\}$.
20. $W_N \leftarrow (P, T_{log}, F)$ ▌
--------------------------------------------------------------------------------------------------------------
```

The algorithm *HR* functions as follows. First, it explores the traces of the event log in step 1. Steps 2 to 7 include the initialization of the input log L^{-DT}, the flag *isDup* to assess whether there are duplicate tasks, the flag *isIdentify* to investigate whether to identify the original input event log L, the flag *isStandardConstructs* to determine the standard constructs in the log, the flag *isComplexConstructs* to assess whether complex constructs exist in the log, and the flag *isOverlap* to identify overlapping constructs from detected constructs. Step 8 is borrowed directly from (Li et al., 2007). In this step, the A/D Table γ of each task is constructed, and each table is examined for duplicate tasks based on rules (R23*), (R24), and (R25) in the function *judgeDuplicate*. Detected duplicate tasks are registered in tuple θ' of γ, the renamed task t' is added in T_{log}, and the new event log is still reserved in L^{-DT}. In step 9, a new A/D Table γ' of each task is constructed, because the original event log is updated when duplicate tasks are detected. In step 10, each table of each task is assessed for the standard constructs (causal dependency, AND-split, AND-join, XOR-split, XOR-join, first tasks, and last tasks) based on definitions 1 to 10, which were introduced in the standard constructs detection section. These definitions are in the function *judgeSTDConst.* Detected standard constructs are identified in β_k. In steps 11 and 12, the first and last tasks detected by the standard construct rules are returned in T_{first} and T_{last}, respectively. Similar to step 10, in step 13, each table of each task is investigated for complex constructs (IvT-SR, IvT-LR, IvT-SS, IvT-LS, IvT-SDB, IvT-SDE, IvT-SW, L_{1p}, L_{2p}, and NFC) based on rules (R1 to R10, R12 to R15, R17 and R19 to R22) in the function *judgeCOMPLEXConst*. Note that in this step, invisible tasks are investigated first before loops and non-free choice constructs. If invisible tasks are detected, they are added to the log and the previous log Log^{-DT} is updated. After that loops and non-free choice constructs are investigated. The detected complex constructs are identified in β'_k. In step 14, the detected standard and complex constructs are saved in D_c. In step 15, the overlapping constructs from the detected constructs are identified by rules (R11), (R16) and (R18) in the function *judgeOverlapping*. Identified overlapping constructs are excluded in step 16. Steps 17 to 20 are directly borrowed from (Van der Aalst et al., 2004). In these steps, the places and the connecting arcs are built, and the discovered model in the workflow net is returned. Note that this ordering of detection is compulsory. Duplicate tasks must be detected first, or all results will be wrong. In addition, standard constructs need to be detected before complex constructs, because complex constructs depend on standard constructs.

Conclusion

In this chapter, we introduced and explained all steps of the heuristic rule-based technique for process model discovery from event logs. For the construction of the Heuristic Rule-based algorithm, a number of notations were designed to mine standard constructs (sequence, AND-split/join, and XOR-split/join), and a number of heuristic rules were developed to detect complex constructs (short loops, invisible tasks, non-free choice constructs, and duplicate tasks). One of the main features of the HR algorithm is that the modularity of the rules typically enables a knowledge base to be easily updated, extended or modified. In the next chapter we will evaluate the algorithm using artificial and real-life data.

Chapter 7 HEURISTIC RULE-BASED ALGORITHM EVALUATION

In this section, we evaluate the HR algorithm using small artificial event logs. Then, we compare the HR algorithm with current algorithms in terms of its ability to mine standard and complex constructs. Finally, we evaluate our technique based on a real-life event log.

7.1. Evaluation based on small artificial logs

Before using real-life models, we first focus on smaller artificial examples that demonstrate the applicability of the HR algorithm. To test our framework, we applied the HR algorithm to 13 complete event logs generated from 13 artificial original models. The original models M_O were manually constructed with 13 levels such that, at each new level, a complex construct was added to the model of the previous level. The more constructs we added to the model, the more complex it became. This way we can evaluate the ability of the algorithm in mining standard and complex constructs when they are incorporated together. Level 1 was a simple model containing a sequence and an exclusive choice. Level 2 was a model with a sequence, choice, and parallelism. At level 3, there was a sequence, choice, parallelism, and a loop of length one. At level 4, a loop of length two was added to the model in level 3. At level 5, an invisible task of type short redo was incorporated with the two short loops, choice, and parallelism. At level 6, the model contained a sequence, choice, parallelism, a long loop, a type of non-free choice construct, and short- and long-skip invisible tasks. The model of level 7 contained a choice, parallelism, a loop of length one, a loop of length two, a short-redo invisible task, and a long-redo invisible task. At level 8, the model included a choice, parallelism, short loops, short- and long-redo invisible tasks, and a short-skip invisible task. In the model of level 9, there was a choice and parallelism, along with short loops, short- and long-redo invisible tasks, and short- and long-skip invisible tasks. For the model of level 10, there were invisible tasks of type skip involved in a non-free choice construct. In the model of level 11, an invisible task of type side in the beginning was incorporated with short- and long-skip invisible tasks, a short-redo invisible task, short loops, choice, and parallelism. At level 12, the model contained an invisible task of type switch, choice, short-and long-skip invisible tasks, short- and long-redo invisible tasks, a loop of length two, and an invisible task of type side at the end. Finally, level 13 included a sequence, choice, parallelism, loops of length one and length two, another type of non-free choice construct, and short- and long-skip invisible tasks. In this set of models, the maximum number of activities in a process model was less than 20, and the number of cases in one event log was less than 10. After applying the algorithm to L_i logs such that $1 \leq i \leq 13$, we obtained M_{D_i} models.

To evaluate the discovered models, we adopted conformance verification using conformance metrics. In this paper, we used three types of metrics: fitness (i.e., *f*), to identify whether the observed behaviour (i.e., the event log) was in compliance with the control flow specified by the process model; behavioural similarity (B_p *and* B_r), to evaluate how similarly the discovered model and the original model behaved in terms of precision and recall; and structural similarity (S_p *and* S_r), to assess how structurally similar the discovered model and the original model were in terms of precision and recall (De Medeiros et al., 2007). The values of $f, B_p, B_r, S_p,$ and S_r can range from 0 to 1. The value of f should be as close to 1 as possible. If $f = 1$, the log can be parsed by the process model without any error. If the values of $B_p, B_r, S_p,$ and S_r are close to 1, the original and discovered models are very similar. We also counted the number of transitions, places, and arcs of models M_D and M_O. The results are illustrated in Table 7.1.

Table 7.1. The performance evaluation results

	Level 1		Level 2		Level 3		Level 4		Level 5		Level 6		Level 7		Level 8		Level 9		Level 10		Level 11		Level 12		Level 13	
B_p	1		1		1		1		1		1		1		1		1		1		1		1		1	
B_r	1		1		1		1		1		1		1		1		1		1		1		1		1	
S_p	1		1		1		1		1		0.91		1		1		1		1		1		1		1	
S_r	1		1		1		1		1		1		1		1		1		1		1		1		1	
	M_O	M_D	M_O	M_D	M_O	M_D	M_O	M_D	M_O	M_D	M_O	M_D	M_O	M_D	M_O	M_D	M_O	M_D	M_O	M_D	M_O	M_D	M_O	M_D	M_O	M_D
$\#T$	5	5	8	8	11	11	12	12	14	14	9	10	17	17	18	18	19	19	7	7	22	22	23	23	17	17
$\#P$	6	6	9	9	10	10	10	10	12	12	9	10	12	12	13	13	13	13	9	9	17	17	17	17	14	14
$\#A$	12	12	18	18	24	24	26	26	30	30	22	24	36	36	38	38	40	40	22	22	48	48	48	48	38	38

As shown in Table 7.1., the value of f is equal to 1 for all levels. Also, the discovered models are similar to the original models at levels 1, 2, 3, 4, 5, 7, 8, 9, 10, 11, 12, and 13, and all complex constructs (short loops, invisible tasks, and non-free choice constructs) are correctly discovered, although they are incorporated with each other. Even when the invisible tasks are in a non-free choice construct, they are correctly detected. The values of

$B_p, B_r, S_p,$ and S_r are 1 at all levels. This demonstrates that the HR algorithm is applicable, except at level 6, where the value of S_r is 0.91. Moreover, the numbers of transitions, places, and arcs of the mined models in all levels are equivalent to those of the original models, except at level 6, where the number of transitions and arcs is greater than in the original model.

At level 6, although the mined model and the original model are slightly structurally different ($S_r = 0.91$), they share the same behaviour (see Figure 7.1). This is because the loop between tasks H and I in the original model has been detected as an invisible task of type redo, which allows similar behaviour. The main outcome is that the implicit dependency between tasks B and D is detected correctly, as well as the IvT-SS.

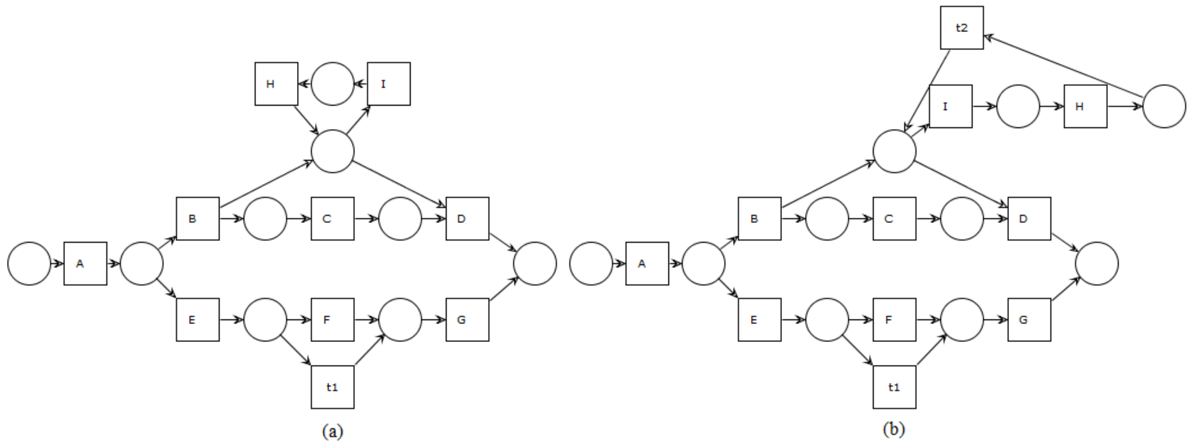


Figure 7.1. The process model of level 6: (a) original model, (b) model mined with the HR algorithm

7.2. Comparison with other algorithms

Let us now apply several algorithms to the 13 complete event logs used in the previous section and also to the complete sample event logs used throughout the paper in order to compare the abilities of our algorithm and current mining algorithms for mining standard and complex constructs. The algorithms we applied were α^{++}, $\alpha^{\#}$, $\alpha^{\$}$, Inductive Miner (IM), HeuristicsMiner (HM), ILP (Werf et al.,2009), ETM (Buijs et al., 2012), Region Miner (RM), Transition System (TS) (Kalenkova et al, 2014), DWS (Greco et al, 2006), and Genetic Miner (GM). We imported the complete event logs with the ProM tool (Van Dongen et al., 2005) and ran the plugin of each of the aforementioned algorithms. We compared the models mined by the different algorithms and our HR algorithm with the original models based on a visual inspection. Table 7.2 illustrates the obtained results. 'Yes' indicates that the construct in the original model was correctly detected in the mined model, 'No' indicates that the construct in the original model was not detected in the mined model, and 'Sb' indicates that the mined construct in the discovered model was not structurally similar to that in the original model but was behaviourally similar.

Table 7.2. Comparison of the ability of current algorithms in mining standard and complex constructs

	HR	α^{++}	$\alpha^{\#}$	$\alpha^{\$}$	IM	HM	ILP	ETM	RM	TS	DWS	GM
SPC	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
$\mathbf{L_{1p}}$	Yes	Yes	Yes	Yes	Sb	No	Yes	Yes	No	Sb	No	Yes
$\mathbf{L_{2p}}$	Yes	Yes	Yes	Yes	Yes	Sb	Yes	No	Yes	Yes	Yes	Yes
$\mathbf{IvT_{SR}}$	Yes	Sb	Yes	Yes	Yes	No	No	No	No	Sb	Yes	No
$\mathbf{IvT_{LR}}$	Yes	No	Yes	Yes	Yes	Yes	No	No	Sb	No	Yes	Yes
$\mathbf{IvT_{SS_{seq}}}$	Yes	No	Yes	Yes	Yes	Yes	No	Yes	No	Yes	Yes	Yes
$\mathbf{IvT_{SS_{par}}}$	Yes	No	No	Yes	Yes	No	No	No	No	Sb	No	No
$\mathbf{IvT_{LS_{seq}}}$	Yes	No	Yes	Yes	Yes	Yes	No	Yes	No	No	Yes	Yes
$\mathbf{IvT_{LS_{par}}}$	Yes	No	No	Yes	Yes	Yes	No	No	No	No	No	Yes
$\mathbf{IvT_{SDB}}$	Yes	No	Yes	Yes	Yes	No	No	Yes	Yes	No	No	No
$\mathbf{IvT_{SDE}}$	Yes	No	Yes	Yes	Yes	No	No	Yes	Yes	No	No	No
$\mathbf{IvT_{SW}}$	Yes	No	Yes	Yes	No	Yes	No	No	No	No	Yes	Yes
$\mathbf{NFC_1}$	Sb	Yes	No	Yes	No	No	Yes	No	Yes	No	No	Sb
$\mathbf{NFC_2}$	Yes	Yes	No	Yes	No	No	No	No	No	Yes	Yes	Yes
$\mathbf{NFC_3}$	Yes	Yes	No	Yes	No	No	Yes	No	Yes	No	No	No
$\boldsymbol{IvT_{in-NFC}}$	Yes	No	No	Yes	No	No	No	No	No	No	No	No
$\mathbf{DT_{N-CE}}$	Yes	No	No	No	No	No	No	Yes	No	No	No	No
$\mathbf{DT_{CE}}$	Yes	No	No	No	No	No	No	No	No	No	No	Yes

'SPC' refers to sequence, parallel, and choice, DT_{N-CE} refers to a duplicate task in the case of non-cross-equivalence, and DT_{CE} refers to a duplicate task in the case of cross-equivalence.

All algorithms have no problem in correctly discovering the standard (sequence, choice, and parallel) constructs. The problem appears in the mining of complex constructs. As can be seen, no algorithm is able to correctly mine all complex constructs. Each algorithm differs in its ability to mine the different complex constructs. For instance, the inductive miner is capable of correctly discovering loops of length two and most invisible task types, except for invisible tasks of type switch, where the algorithm fails. It can discover behaviour similar to that of loops of length one, but it is not capable of discovering non-free choice constructs or duplicate tasks. Similarly, the other algorithms are capable of mining some constructs but are unable to discover other constructs. With the same complete event logs, our HR algorithm is able to correctly detect short loops, invisible tasks, duplicate tasks, and non-free choice constructs.

7.3. Evaluation based on a real-life log

The above evaluation results demonstrate that our HR algorithm is applicable to artificial data. Now, we use a more realistic example, depicted in Figure 7.2., to demonstrate the applicability of the HR algorithm. The process model shown in Figure 7.2 was mined based on a log obtained from the IEEE TF on Process Mining - Event Logs. The events of this real-life event log were sepsis cases from a hospital. Sepsis is a life-threatening condition that appears because of an infection. One case depicts the pathway through the hospital. The events were recorded by the ERP (Enterprise Resource Planning) system of the hospital. We filtered incomplete cases and noisy data in the event log with the ProM tool plugin ‘Filter using simple heuristics’. The filtered log contains 973 events, 155 cases, and 7 activities.

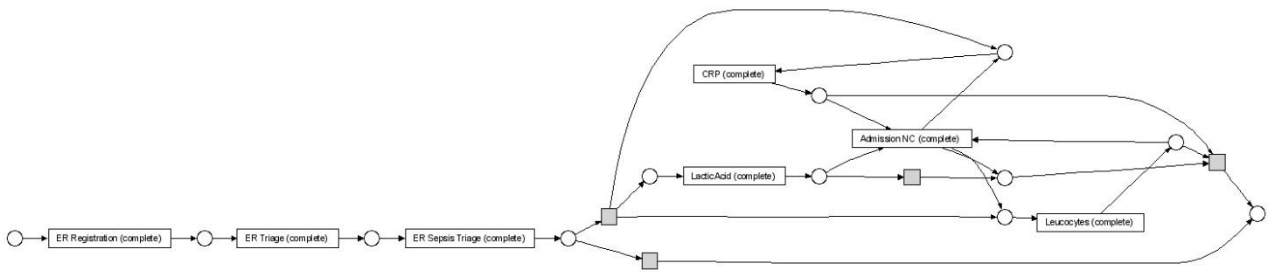
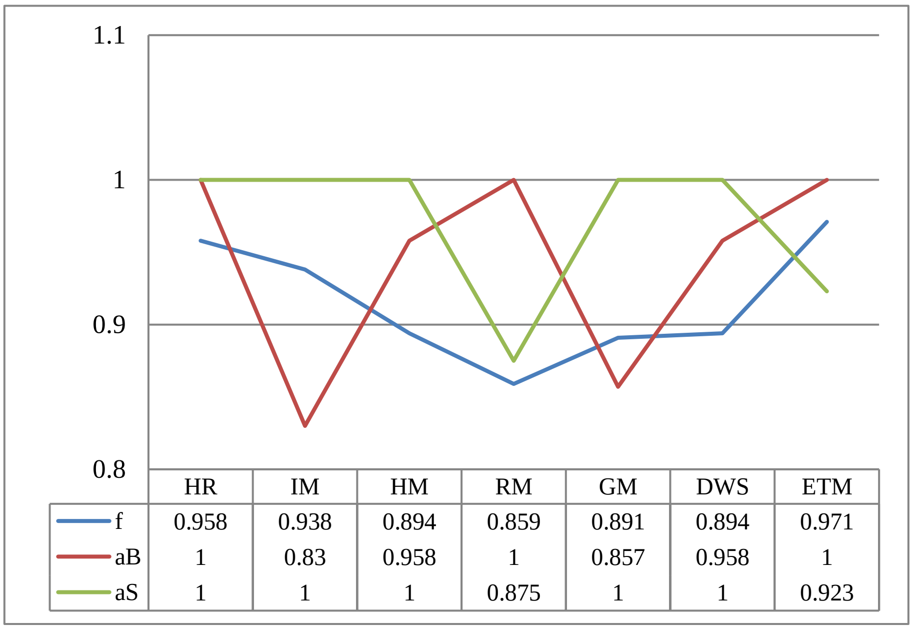


Figure 7.2. Workflow net model discovered by the HR algorithm from a real-life event log

By following the steps of our algorithm, we discovered the process model shown in Figure 7.2 by hand, drew the obtained model with the WoPeD tool, saved the model into a PNML extension, and then imported the obtained model and the real-life log with the ProM tool to verify the conformance of the model. Since, in reality, the original model is often unknown and only the log is available, we did not use the behavioural similarity and structural similarity metrics that we used to evaluate artificial data. Instead, we used the dimension of appropriateness for conformance verification, as well as fitness.

Appropriateness is used to determine whether the observed process (i.e., event log) is described in a suitable way by the model. In this evaluation, we used three metrics: *fitness* (*f*), *behavioural appropriateness* (aB), and *structural appropriateness* (aS), as introduced in (Rozinat and van der Aalst, 2006). *Behavioural appropriateness* identifies the degree of accuracy with which the process model describes the observed behaviour, while *structural appropriateness* identifies the degree of clarity with which the process model is represented. *Fitness* is defined in the previous section. Only if f is close to 1 can aS and aB be considered. If the fitness is good, higher values of aS and aB are desirable. To compare the quality of the model discovered by our HR algorithm with other algorithms, we applied the α^{++}algorithm, $\alpha^{\#}$ algorithm, Inductive Miner (IM), HeuristicsMiner (HM), ILP, ETM, Region Miner (RM), Transition System (TS), DWS, and Genetic Miner (GM) to the same real-life event log. The results are illustrated in Figure 7.3. Note that only the algorithms available in ProM are applied.



	HR	IM	HM	RM	GM	DWS	ETM
f	0.958	0.938	0.894	0.859	0.891	0.894	0.971
aB	1	0.83	0.958	1	0.857	0.958	1
aS	1	1	1	0.875	1	1	0.923

Figure 7.3. Comparison of mining results by the HR algorithm and current algorithms using a complete real-life event log . **f** is the fitness value of the model, **aB** refers to behavioral apappropriateness, and **aS** denotes structural appropriateness.

Very low fitness values were obtained for the models discovered by the α^{++}algorithm, $\alpha^{\#}$ algorithm, ILP, and Transition System. Therefore, we could not compute their behavioural and structural appropriateness. Only the algorithms with good fitness are displayed in Figure 7.3. As can be seen, the mining results for our HR algorithm are impressive and better than those for the other algorithms. The fitness of the model discovered by the HR algorithm is greater than the fitness of the models mined by the IM, HM, RM, GM, and DWS algorithms. Only the model mined by ETM has a slightly higher fitness value than the model derived by the HR algorithm. Moreover, the values of aB and aS for conformance between the complete real-life log and the mined model are greater for the HR algorithm than for the IM, HM, RM, GM, DWS, and ETM algorithms. The model mined by the HR algorithm is the only model in which the value of fitness is very good and the values of aB and aS are both equal to 1.

7.4.Limitations of the HR algorithm

Despite the ability of the HR algorithm to successfully mine standard (sequence, choice, and parallel) constructs and complex constructs (short loops, invisible tasks, duplicate tasks, and non-free choice constructs), there are still some sound workflow nets that cannot be mined by the HR algorithm (e.g., the workflow net *N* shown in Figure 7.4). One complete event log of *N* is {<A,B,C,E>, <A,C,B,E>, <A,B,D,D,C,E>}. Using this event log as input, the HR algorithm discovers the model *N'* shown in Figure 7.4. After *A* is executed, a token will be produced in p7 and p9. If *B* is executed, there will be a token in p8. If *D* is executed twice, there will be a token in p8 and two tokens in *C*. Then, if *C* is executed, the workflow will terminate, but one token will remain in p9. This is because the dependency between *A* and *D* is not detected; thus, the model is not sound. Perhaps more advanced heuristic rules introduced in the future will be able to handle this case. The HR algorithm is based on heuristic rules and one of the principal features of rule-based algorithms is that the modularity of the rules typically enables a knowledge base to be easily updated, extended or modified.

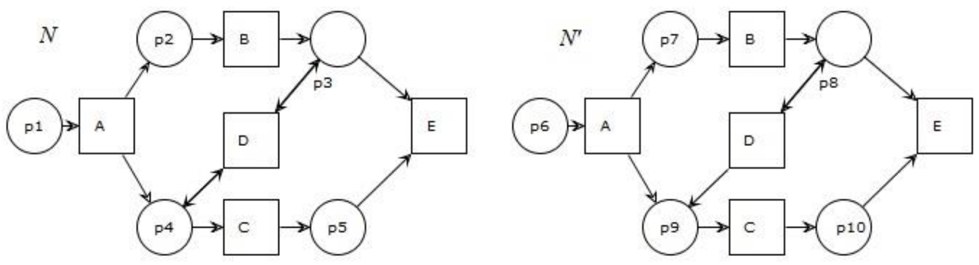


Figure 7.4. N is the reference model, and N' is the model mined by the HR algorithm

More basic issues from a practical point of view are problems related to completeness and noise. The correctness of the HR algorithm is based on the assumption that a given event log is complete. Therefore, it is impossible to mine a behaviour that has not yet been recorded in the log due to a short period of observation. Actually, this problem is not so much an issue for the HR algorithm but is a major problem in process mining.

The other issue is noise. The HR algorithm operates on the assumption that all events recorded in the event log are correct, but this is not the case in the real world. In several works, the assumption that the log is noise free is considered as a limitation. However, this should not be considered as a limitation, since several techniques have been proposed to filter noise before applying the process discovery algorithms. The event log filtering techniques that have been proposed to address the problem of noise can typically be classified into four categories: 1) techniques for filtering events (e.g. Conforti et al, 2017), 2) techniques for filtering traces (e.g., Ghionna et al., 2008), 3) techniques for filtering activities (e.g., Tax et al., 2017), and 4) filtering techniques integrated in the process discovery algorithm (e.g., Leemans et al., 2013). The first, second and third category of log filtering techniques are proposed to be used by the algorithms operating on the assumption that the log contains correct information. These noise filtering techniques can be used before applying the Heuristic Rule-based algorithm.

Process discovery offers a new paradigm for process modelling based on recorded events. The discovered models provide organisations with a full understanding of how their processes are executed and what is really happening in the organisation. This enables them with full knowledge to make better business decisions. Although plenty of process discovery techniques have been developed and applied successfully, no algorithm is capable of discovering all structures from the event log in a restricted time. In the previous chapter, we proposed a process discovery approach, the Heuristic Rule-based algorithm, in which a number of notations were designed to mine standard constructs (sequence, AND-split/join, and XOR-split/join), and a number of heuristic rules were developed to detect complex constructs (short loops, invisible tasks, non-free choice constructs, and duplicate tasks). One of the main features of the HR algorithm is that the modularity of the rules typically enables a knowledge base to be easily updated, extended or modified.

In this chapter, the approach is evaluated with both artificial and real-life data. The outcomes show that the heuristic rules can correctly detect the aforementioned constructs even when they are involved together. We believe that this work provides a full understanding of the complex constructs which can help other researchers to extend or improve the existing algorithms in terms of mining complex constructs.

Future work can mainly focus on the following points. First, investigating other cases and extending or improving the heuristic rules. Then, implementing the approach with larger real-life data sets.

Chapter 8 CONCLUSION

Process mining is a set of techniques whereby knowledge from event log stored in today's information systems are extracted to automatically construct business process models to have a full understanding of the real behaviour of processes, identify bottlenecks, and then improve them. In this thesis, we presented an overview of process mining framework. We presented a critical review of the state of the art of process mining challenges. We described an industrial application of process mining to demonstrate the applicability and show the importance of process mining in handling real-life problems. The critical review of process mining challenges has shown that there are many process discovery algorithms that have been proposed today to represent the real behaviour of a business process, and that there is a lack of a benchmark tool or a recommendation framework that recommend a suitable algorithm to a given event log. There are actually complex control-flow constructs that current discovery techniques cannot correctly discover in models based on event logs. These constructs are short loops, invisible tasks, and non-free choice constructs. There is currently no algorithm that can handle all of these structures in a restricted time. Therefore, we proposed a framework that recommends the process discovery algorithm appropriate to a given event log. Recommendation framework consists first of detecting the existing complex control-flow constructs in the event log without discovering any model, then recommending the algorithm suitable to the given event log based on a knowledge database containing information regarding the ability of each process discovery algorithm in mining the complex constructs, based on the computation time, and based on the ability of algorithms in mining sound models. We have evaluated the recommendation framework using artificial and real-life event log. The results show that the framework can recommend correctly suitable algorithms to a given event log. This framework doesn't consider duplicate tasks. Therefore, including the detection of duplicates tasks in the implementation will be considered in the future work.

Since each of the existing discovery algorithms can handle specific complex constructs while cannot handle some constructs, we introduced a new heuristic rule-based algorithm that is capable of handling all constructs: shorts loops, non-free choice constructs, invisible tasks and duplicate tasks. For the construction of the Heuristic Rule-based algorithm, a number of notations were designed to mine the standard constructs (sequence, AND-split/join, and XOR-split/join), and a number of heuristic rules were developed to discover the complex constructs (short loops, invisible tasks, non-free choice constructs, and duplicate tasks). One of the main features of the HR algorithm is that the modularity of the rules typically enables a knowledge base to be easily updated, extended or modified. The approach was evaluated with both artificial and real-life data. The outcomes show that the heuristic rules can correctly detect the aforementioned constructs even when they are involved together. Future work can mainly focus on the following points. First, investigating other cases and extending or improving the heuristic rules, then, implementing the approach with larger real-life data sets.

REFERENCES

Adriansyah, A. (2014). Aligning Observed and Modeled Behavior. PhD's Thesis. Eindhoven University of Technology.

Adriansyah, A., Dongen, B.F., Piessens, D., Wynn, M.T., and Adams, M. (2012)' Robust Performance Analysis on YAWL Process Models with Advanced Constructs*', Journal of Information Technology Theory and Application,* Vol.12 No.3, pp.5-26.

Aguirre, S., Parra, C., and Alvarado, J. (2013)' Combination of Process Mining and Simulation Techniques for Business Process Redesign: A Methodological Approach' *International Federation for Information Processing 2013, Data-Driven Process Discovery and Analysis, Lecture Notes in Business Information Processing,* Springer-Verlag Berlin Heidelberg, pp.24-43.

Aksu, U., Schunselaar, D. M. M., Reijers, H. A. (2016)'A Cross-Organizational Process Mining Framework for Obtaining Insights from Software Products: Accurate Comparison Challenge' *IEEE 18th Conference on Business Informatics (CBI),* pp.2378-1971, Paris.

Aruna Devi, C., and Sudhamani. (2012). Process Mining In Corrugated Boxes Manufacturing Industry - A Case Study. *International Journal of Emerging Trends & Technology in Computer Science*, 1(3).

Aruna, P., and Laxmi Priya, P. (2015)' Dealing With Concept Drifts in Process Mining' *International Journal & Magazine of Engineering, Technology, Management and Research,* Vol.2 No.8, pp.2348-4845.

Bergenthum R., Desel J., Lorenz R., and Mauser S. (2007) 'Process mining based on regions of languages' in *Lecture Notes in Computer Science 4714*, Springer, Berlin, Heidelberg, pp. 375-383.

Bolt, A., Leoni, M.D., Aalst, Wil M. P., and Gorissen, P. (2015)' Exploiting Process Cubes, Analytic Workflows and Process Mining for Business Process Reporting: A Case Study in Education' *International Symposium on Data-driven Process Discovery and Analysis,* Vienna, Austria, pp.33-47.

Bose, R. P. J. C., Aalst, W.M.P., Zliobaite, I., and Pechenizkiy, M. (2014)' Dealing with Concept Drifts in Process Mining: A Case Study in a Dutch Municipality' *IEEE Transactions on Neural Networks and Learning Systems,* Vol. 25 No.1, pp.154-171.

Bose, R.P.J.C., and Aalst, W.M.P. (2013)' Discovering Signature Patterns from Event Logs' Hammer, B., Zhou, Z.H., Wang, L., & Chawla, N. (eds.), *IEEE Symposium on Computational Intelligence and Data Mining,* Singapore, pp.111-118.

Buijs J.C.A.M., van Dongen B.F., and van der Aalst W.M.P. (2012) 'On the Role of Fitness, Precision, Generalization and Simplicity in Process Discovery', in *On the Move to Meaningful Internet Systems: OTM 2012*, Springer, pp. 305–322.

Buijs, J.C.A.M. (2014a) *Flexible Evolutionary Algorithms for Mining Structured Process Models*. Unpublished Ph.D. Thesis, Eindhoven University of Technology, Netherland.

Buijs, J.C.A.M., and Reijers, H.A. (2014b)' Comparing Business Process Variants Using Models and Event Logs' *Enterprise, Business-Process and Information Systems Modeling, Lecture Notes in Business Information Processing,* Vol. 175, pp.154-168, Springer-Verlag Berlin Heidelberg.

Buijs, J.C.A.M., Dongen, B.F., and Aalst, W.M.P. (2012a)' A genetic algorithm for discovering process trees' *IEEE World Congress on Evolutionary Computation,* Brisbane, Australia, pp.1-8.

Buijs, J.C.A.M., Dongen, B.F., and Aalst, W.M.P. (2012b)' Towards Cross-Organisational Process Mining in Collections of Process Models and their Executions' *Business Process Management Workshops,* pp.2-13.

Buijs, J.C.A.M., Dongen, B.F., and Aalst, W.M.P. (2013a)' Quality Dimensions in Process Discovery: The Importance of Fitness, Precision, Generalisation and Simplicity' *International Journal of Cooperative Information Systems,* Vol. 23 No.1, pp.1-39.

Buijs, J.C.A.M., Dongen, B.F., and Aalst, W.M.P. (2013b)' Mining Configurable Process Models from Collections of Event Logs' *Proceedings of the 11th International Conference on Business Process Management, Lecture Notes in Computer Science,* Beijing, China, Springer-Verlag Berlin Heidelberg, pp.33-48.

Burattin, A., Conti, M., and Turato, D. (2015) 'Toward an anonymous process mining' *Proceedings of the 3rd International Conference on Future Internet of Things and Cloud (FiCloud),* pp. 58–63.

Calvanese D., Montali M., Syamsiyah A., van der Aalst W.M.P. (2016) 'Ontology-Driven Extraction of Event Logs from Relational Databases' *Reichert, M., & Reijers H. (eds)*

Business Process Management Workshops. BPM, Lecture Notes in Business Information Processing, Vol. 256. Springer, Cham.

Cheng, H-J., and Kumar, A. (2015)' Process Mining on Noisy Logs – Can log sanitisation help to improve performance?' *Decision Support Systems,* Vol.79, pp.138-149.

Cho, M., Song, M., and Yoo, S. (2015). A Systematic Methodology for Outpatient Process Analysis based on Process Mining. *International Journal of Industrial Engineering: Theory, Applications and Practice,* 22(4).

Claes, J. and Poels, G. (2014) 'Merging event logs for process mining: A rule based merging method and rule suggestion algorithm' *Expert Systems with Applications,* Vol.41, pp.7291–7306.

Conforti R., La Rosa M., and ter Hofstede A.H.M (2017) 'Filtering out infrequent behavior from business process event logs', *IEEE Transactions on Knowledge and Data Engineering,* Vol. 29 No.2, pp.300–314.

Conforti, R., La Rosa, M., and ter Hofstede, A.H.M. (2017) 'Filtering Out Infrequent Behavior from Business Process Event Logs' *IEEE Transactions on Knowledge and Data Engineering,* Vol. 29, pp. 300-314.

Conforti, R., Leoni, M., La Rosa, M., and Aalst, W. M. P. (2013)' Supporting Risk-Informed Decisions during Business Process Execution' in *CAiSE 2013: Proceedings of the 25th International Conference, Advanced Information Systems Engineering, Lecture Notes in Computer Science*, pp.116-132, Valencia, Spain, Springer-Verlag Berlin Heidelberg.

De Medeiros A.K.A, Weijters A.J.M.M, and van der Aalst W.M.P. (2007) 'Genetic process mining: an experimental evaluation', *Data Mining and Knowledge Discovery*, Vol. 14 No.2, pp. 245–304.

De Medeiros, A.K.A, van Dongen B.F, van der Aalst W.M.P, and Weijters A.J.M.M. (2004) 'Process mining: extending the alpha-algorithm to mine short loops', *BETA Working Paper Series, WP 113*, Eindhoven University of Technology, Eindhoven.

De Murillas E.G.L., van der Aalst W.M.P., Reijers H.A. (2015) 'Process Mining on Databases: Unearthing Historical Data from Redo Logs' *Motahari-Nezhad H., Recker J., & Weidlich M. (eds), Business Process Management, Lecture Notes in Computer Science*, Vol. 9253. Springer, Cham.

Era, M., Astutia, H. M., and Nastiti, A. (2015). Analysis of Customer Fulfilment with Process Mining: A Case Study in a Telecommunication Company. *Proceeding of the 3rd International Conference on Information Systems, Procedia Computer Science,*72: 588 – 596.

Evermann, J., Thaler, T., and Fettke, P. (2015)' Clustering Traces using Sequence Alignment' *Proceedings of the 11th International Workshop on Business Process Intelligence,* Innsbruck, Austria, pp.11

Fahland, D., and Aalst, W. M. P. (2012)' Simplifying Discovered Process Models in a Controlled Manner' *Information Systems,* Vol. 38 No.4, pp.585-605.

Fernandez-Llatas, C., Lizondo, A., Monton, E., Benedi, J-M., and Traver, V. (2015)' Process Mining Methodology for Health Process Tracking Using Real-Time Indoor Location Systems' *Sensors,* Vol.15 No.12, pp.29821-29840.

Ghionna L., Greco G., Guzzo A., and Pontieri L. (2008) 'Outlier detection techniques for process mining applications', in *Proceeding of the International Symposium on Methodologies for Intelligent Systems*, Springer, pp 150–159.

Goedertier, S., De Weerdt, J., Martens, D., Vanthienen, J., and Baesens, B. (2011). Process discovery in event logs: an application in the telecom industry. *Applied Soft Computing*, 11 (2):1697-1710.

Goedertier, S., Martens, D., Vanthienen, J., and Baesens, B. (2009). Robust Process Discovery with Artificial Negative Events. *Journal of Machine Learning Research*, 10:1305-1340.

Greco G., Guzzo A., Pontieri L., and Saccà D. (2006) 'Discovering Expressive Process Models by Clustering Log Traces', *IEEE Transactions on Knowledge and Data Engineering,* Vol. 18 No.8, pp.1010–1027.

Gunther, C., and Rozinat, A. (2012) 'Disco: Discover your processes' *BPM (Demos). CEUR Work- shop Proceedings,* Vol. 940, pp. 40–44. CEUR-WS.org

Gunther, C., and Rozinat, A. (2012). Disco: Discover your processes. *CEUR Work- shop Proc*. vol. 940 pp. 40–44. Proceedings of CEUR Workshop, 940:40-44.

Günther, C., and van der Aalst, W. M. P. (2007). Fuzzy Mining – Adaptive Process Simplification Based on Multi-perspective Metrics. *International Conference on Business Process Management, LNCS,*4714:328-343, Alonso, G., Dadam P., & Rosemann, M. (eds), Springer-Verlag Berlin Heidelberg.

Guo Q, Wen L, Wang J, Yan Z, Yu P S. (2015) 'Mining invisible tasks in non-free-choice constructs', in *Lecture Notes in Computer Science 9253*, Motahari-Nezhad ., Recker J, Weidlich M (eds.), Springer, Cham, pp. 109–125.

Gupta, E.P. (2014)' Process Mining A Comparative Study' *International Journal of Advanced Research in Computer and Communication Engineering,* Vol. 3 No.11, pp.278-1021.

Hompes, B., Verbeek, E., and Aalst, W.M.P. (2014)' Finding Suitable Activity Clusters for Decomposed Process Discovery' *Accorsi, R., Ceravolo, P., & Russo, B. (eds.), Proceedings of the 4th International Symposium on Data-driven Process Discovery and Analysis, Lecture Notes in Business Information Processing,* pp.32-57, Springer International Publishing.

Hompes, B.F.A., Buijs, J.C.A.M., Van der Aalst, W.M.P., Dixit, P.M., and Buurman, J. (2015a)' Discovering Deviating Cases and Process Variants Using Trace Clustering' *In Proceedings of the 27th Benelux Conference on Artificial Intelligence (BNAIC),* Hasselt, Belgium.

Hompes, B.F.A., Buijs, J.C.A.M., van der Aalst, W.M.P., Dixit, P.M. and J. Buurman. (2015b)' Detecting Change in Processes Using Comparative Trace Clustering' *International Symposium on Data-driven Process Discovery and Analysis,* Vienna, Austria.

IEEE Task Force on Process Mining. (2011)' Process Mining Manifesto' *BPM Workshops, Lecture Notes in Business Information Processing*, Vol. 99, pp. 169–194, Springer-Verlag.

Kakkad, T. and Sheikh, R. (2016) 'Study of Detecting and Localizing Concept Drifts from Event Logs in Process Mining' *International Journal of Advanced Research in Computer and Communication Engineering,* Vol. 5 No.9.

Kalenkova A.A., Lomazova I.A., van der Aalst W.M.P. (2014) 'Process Model Discovery: A Method Based on Transition System Decomposition', in: Ciardo G., Kindler E. (eds) *Application and Theory of Petri Nets and Concurrency, Lecture Notes in Computer Science*, vol 8489. Springer, Cham.

Kalenkova, A., Leoni, M., and Aalst, W.M.P. (2014a)' Discovering, Analyzing and Enhancing BPMN Models Using ProM' Limonad, L., & Weber, B. (eds.), *Business Process Management Demo Sessions, CEUR Workshop Proceedings,* Vol.1295, pp.36-40.

Kalenkova, A., Lomazova, I.A., and Aalst, W.M.P. (2014b)' Process Model Discovery: A Method Based on Transition System Decomposition' *Ciardo, G., & Kindler, E. (eds.),* *Applications and Theory of Petri Nets, Lecture Notes in Computer,* pp.71-90, Springer-Verlag Berlin Heidelberg.

Kalenkova, A.A., Aalst, W. M. P., Lomazova, I.A., and Vladimir, A.R. (2015)' Process Mining Using BPMN: Relating Event Logs and Process Models' *Software & Systems Modeling,* pp.1-30.

Lawal, N.T. A., Odeniyi, O. A., and Kayode, A. A. (2015) 'Application of Data Mining and Knowledge Management for Business Improvement: An Exploratory Study', *International Journal of Applied Information Systems*, Vol. 8 No.3, pp.13-19.

Leemans S J J, Fahland D, van der Aalst W M P. (2013) 'Discovering block-structured process models from event logs - a constructive approach', in *Lecture Notes in Computer Science* 7927, Springer, Berlin, Heidelberg, pp. 311-329.

Leemans, M. and van der Aalst, W.M.P. (2015) 'Process mining in software systems: Discovering real-life business transactions and process models from distributed systems' *ACM/IEEE 18th International Conference on Model Driven Engineering Languages and Systems (MODELS), IEEE,* pp. 44–53.

Leemans, M., and Aalst, W.M.P. (2014)' Discovery of Frequent Episodes in Event Logs' *Accorsi, R., Ceravolo, P., & Russo, B. (eds.), Proceedings of the 4th International Symposium on Data-driven Process Discovery and Analysis, CEUR Workshop Proceedings,* Vol.1293, pp.31-45.

Leemans, S., Fahland, D., and Aalst, W.M.P. (2015b)' Exploring Processes and Deviations' *Fournier, F., & Mendling, J. (eds), Business Process Management Workshops, International Workshop on Business Process Intelligence, Lecture Notes in Business Information Processing,* pp.304-316, Springer-Verlag Berlin Heidelberg.

Leemans, S., Fahland, D., and van der Aalst, W. M. P. (2014). Discovering block-structured process models from event logs containing infrequent behaviour. *International Conference on Business Process Management, LNBIP,* 171:66–78, Springer Cham.

Leemans, S.J.J., Fahland D., van der Aalst W.M.P. (2013) Discovering Block-Structured Process Models from Event Logs - A Constructive Approach. *Application and Theory of Petri Nets and Concurrency, LNCS*, 7927, Colom, J.M., Desel, J. (eds), Springer, Berlin, Heidelberg.

Leemans, S.J.J., Fahland, D., and Aalst, W.M.P. (2014a)' Discovering Block-Structured Process Models from Incomplete Event Logs' *Proceedings of Application and Theory* *of Petri Nets and Concurrency, Lecture Notes in Computer Science,* Tunis, Tunisia, Vol.8489, pp.91-110.

Leemans, S.J.J., Fahland, D., and Aalst, W.M.P. (2015a)' Scalable Process Discovery with Guarantees' *Business Process Modeling, Development, and Support working conference,* Stockholm, Sweden, Vol.214, pp.85-101.

Leemans, S.J.J., Fahland, D., and van der Aalst, W.M.P. (2014b) 'Discovering Block-Structured Process Models from Event Logs Containing Infrequent Behaviour' *Lohmann N., Song M., Wohed P. (eds), Business Process Management Workshops, Lecture Notes in Business Information Processing,* Vol. 171. Springer, Cham.

Leoni, M., and Aalst, W.M.P. (2014)' The FeaturePrediction Package in ProM: Correlating Business Process Characteristics' *Limonad, L., & Weber, B. (eds.), Business Process Management Demo Sessions,* Vol.1295, pp.26-30.

Leoni, M., Maggi, F.M. and Aalst, W.M.P. (2015b)' An Alignment-Based Framework to Check the Conformance of Declarative Process Models and to Preprocess Event-Log Data, *Information Systems,* Vol.47, pp.258-277.

Leoni, M., Suriadi, S., Hofstede, A.H.M., and Aalst, W.M.P. (2015a)' Turning Event Logs Into Process Movies: Animating What Has Really Happened' *Software and Systems Modeling,* pp.1-26.

Leoni, M., Suriadi, S., ter Hofstede, A.H.M. and van der Aalst, W.M.P. (2016b)' Turning event logs into process movies: animating what has really happened' *Software & Systems Modeling,* Vol. 15 No. 3, pp.707. doi:10.1007/s10270-014-0432-2.

Leoni, M., van der Aalst, W.M.P. and Dees, M. (2016a) 'A General Process Mining Framework for Correlating, Predicting and Clustering Dynamic Behavior Based on Event Logs' *Information Systems,* Vol.56, pp.235-257.

Li J, Liu D, and Yang B. (2007) 'Process Mining: Extending α-Algorithm to Mine Duplicate Tasks in Process Logs', in *Lecture Notes in Computer Science 4537,* Springer, Berlin, Heidelberg, pp. 396-407.

Li, H., and Kang, Q. (2015)' A new process reconstruct approach based on the Markov transition matrix of event log' *Journal of Chemical and Pharmaceutical Research, Vol.*7 No.3, pp.1578-1586.

Lu, X., Fahland, D. and van der Aalst, W.M.P. (2016a) 'Interactively Exploring Logs and

Mining Models with Clustering, Filtering, and Relabeling' Accepted as Tool Demo at BPM 2016, Rio de Janeiro, Brasil.

Lu, X., Fahland, D., van den Biggelaar, F. and van der Aalst, W.M.P. (2016b) 'Detecting deviating behaviors without models' *Reichert, M. & Reijers, H.A. (eds), Business Process Management Workshops, Lecture Notes in Business Information Processing*, Vol.256, pp.126-139, Springer.

Ly, L.T., Indiono, C., Mangler, J., and Rinderle-Ma, S. (2012)' Data Transformation and Semantic Log Purging for Process Mining?' *Advanced Information Systems Engineering, Proceedings of 24th International Conference,* Gdansk, Poland, pp.238-253.

Maggi, F.M., Bose, R.P.J.C., and Aalst, W.M.P. (2012b)' Efficient Discovery of Understandable Declarative Process Models from Event Logs' *Proceedings of the 24th international conference on Advanced Information Systems Engineering,* pp.270-285.

Maggi, F.M., Westergaard, M., Montali, M., and Aalst, W.M.P. (2012a)' Runtime Verification of LTL-Based Declarative Process Models' *Runtime Verification, Lecture Notes in Computer Science,* Vol.7186, pp.131-146, Springer-Verlag Berlin Heidelberg.

Mannhardt, F., de Leoni M., Reijers H.A., van der Aalst W.M.P. and Toussaint P.J. (2016) 'From Low-Level Events to Activities - A Pattern-Based Approach' *La Rosa M., Loos P., & Pastor O. (eds), Business Process Management, Lecture Notes in Computer Science,* Vol. 9850. Springer, Cham.

Mannhardt, F., Leoni, M., and Reijers, H.A. (2014)' Extending Process Logs with Events from Supplementary Sources' *Data- & Artifact-centric BPM,* Haifa, Israel.

Mans, R., Aalst, W.M.P., and Verbeek, E. (2014)' Supporting Process Mining Workflows with RapidProM' *Limonad, L., & Weber, B. (eds.), Business Process Management Demo Sessions, CEUR Workshop Proceedings,* Vol.1295, pp.56-60.

Mokhov, A., and Carmona, J. (2014) *Process Mining using Parameterized Graphs,* Newcastle University, NCL-EEE-MICRO-MEMO-2014-009.

Müller, R., Stahl, C., Aalst, W.M.P., and Westergaard, M. (2013)' Service Discovery from Observed Behavior While Guaranteeing Deadlock Freedom in Collaborations' *Proceedings of 11th International Conference on Service Oriented Computing, Lecture Notes in Computer Science,* Berlin, Germany, Vol.8274, pp.358-373.

Munoz-Gama, J., Carmona, J., and Aalst, W.M.P. (2013)' Conformance Checking in the Large: Partitioning and Topology' Daniel, F., Wang, J., & Weber, B. (eds.), *International Conference on Business Process Management, Lecture Notes in Computer Science,* Vol.8094, pp.130-145.

Munoz-Gama, J., Carmona, J., and Aalst, W.M.P. (2014)' Single-Entry Single-Exit Decomposed Conformance Checking. *Information Systems,* Vol.46, pp.102-122.

Nakatumba, J., Westergaard, M., and Aalst, W. M. P. (2012a)' Generating Event Logs with Workload-Dependent Speeds from Simulation Models' *Advanced Information Systems Engineering Workshops, Lecture Notes in Business Information Processing,* Vol.112, pp.383-397.

Nakatumba, J., Westergaard, M., and Aalst, W. M. P. (2012b) *A Meta-model for Operational Support,* BPM Center Report, BPM-12-05.

Nithya, S., Nithya Subha, G., Uma, M., and Pradipa, R. (2015)' Identification of agent-guilt using drift concept in process mining' *International Journal of Research in Computer Applications and Robotics,* Vol.3 No.3, pp.213-218.

Nooijen, E. H. J., Dongen, B. F., and Fahland, D. (2012)' Automatic Discovery of Data-Centric and Artifact-Centric Processes' *The First International Workshop on Data- and Artifact-centric BPM,* Tallinn, Estonia.

Omair, B., and Emam, A. (2015)' Towards Big Business Process Mining' *The First International Conference on Big Data, Small Data, Linked Data and Open Data,* ISBN: 978-1-61208-445-9.

Park, M., Song, M., Baek, T. H., Son, S., Ha, S. J., and Cho, S. (2015). Workload and delay analysis in manufacturing process using process mining. *Proceeding of the 3rd Asia Pacific Conference on Asia Pacific Business Process Management, LNBIP*, 219:138–151, Springer Cham.

Pérez-Alfonso D., Fundora-Ramírez O., Lazo-Cortés M.S., and Roche-Escobar R. (2015) 'Recommendation of Process Discovery Algorithms Through Event Log Classification' Carrasco-Ochoa J., Martínez-Trinidad, J., Sossa-Azuela J., Olvera López J., & Famili F. (eds) *Pattern Recognition, MCPR, Lecture Notes in Computer Science*, Vol. 9116. Springer, Cham.

Pileggi, P., Rivero-Rodriquez, A., and Nykänen, O. (2015)' Towards Traditional Simulation Models of Context Using Process Mining' *7th International Conference on Computational Intelligence, Communication Systems and Networks,* pp.70-75.

Polyvyanyy, A., Vanhatalo, J., V¨olzer, H. (2010)' Simplified computation and generalisation of the refined process structure tree' *Bravetti, M., Bultan, T., (eds.), WS-FM, Lecture Notes in Computer Science,* Vol.6551, pp.25–41, Springer.

Priyadharshini, V., and Malathi, A. (2014)' An efficient process mining model for Petri Nets in process discovery' *IOSR Journal of Computer Engineering,* Vol.164 No.4, pp.65-68, *International Journal of Engineering Research and Applications,* Vol.4 No.2, pp.347-349

Process Mining Wiki. Publications. [online] http://www.processmining.org/publications/start./Accessed 15 September 2016.

Przybylek. M.R. (2015)' Skeletal Algorithms: Sequential Pattern Mining' *International Journal of Computer Theory and Engineering,* Vol.7 No.2, pp.132-138.

Ramezani, E., Fahland, D., and Aalst, W.M.P. (2013)' Supporting Domain Experts to Select and Configure Precise Compliance Rules' *Lohmann, N., Song, M., & Wohed, P. (eds.), Business Process Management Workshops on Security in Business Processes,* Vol.171, pp.498-512.

Raviteja, N.V, Pochiraju, and Kumar, P.P. (2015)' Drift in process mining for analyzing process changes based on event logs' *International Journal of Professional Engineering Studies,* Vol.5 No.5, pp.105-110.

Rekhadevi, B., and Appini, N.R. (2015)' Concept Drifts Characterisation and Detection in Process Mining' *International Journal of Advance Research in Computer Science and Management Studies, Vol.*3 No.9, pp.2321-7782.

Ribeiro, J. And Carmouna, J. (2014)' RS4PD: A Tool for Recommending Control-Flow Algorithms' *BPM (Demos),* pp.66.

Rojas, E., Munoz-Gama, J., Sepúlveda, M., and Capurro, D. (2016). Process Mining in Healthcare: A literature review. *Journal of Biomedical Informatics,* doi: http://dx.doi.org/10.1016/j.jbi.2016.04.007.

Rozinat A., and van der Aalst W.M.P. (2006) 'Conformance Testing: Measuring the Fit and Appropriateness of Event Logs and Process Models', in *Lecture Notes in Computer Science 3812,* Bussler C et al. (eds.), Springer-Verlag, Berlin, pp.163–176.

Rozinat, A., Alves de Medeiros, A.K., Günther, C.W., Weijters, A.J.M.M., and Van der Aalst, W.M.P. (2007) *Toward an Evaluation Framework for Process Mining Algorithms,* BPM Center Report BPM-07-06, BPMcenter.org

Rozinat, A., and van der Aalst, W. M. P. (2008). Conformance Checking of Processes Based on Monitoring Real Behavior. *Information Systems,* 33:64-95.

Rozinat, A., de Jong, I. S. M., Gu¨nther, C. W., and van der Aalst, W. M. P. (2009). Process mining applied to the test process of wafer scanners in ASML. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews),* 39:474-479.

Samalik, J., Kusters, R.J., Trienekens, J.J.M., and Weijters, A.J.M.M. (2013)' Data From Configuration Management Tools As Sources For Software Process Mining' *Proceeding of the Eighth International Conference on Software Engineering Advances,* Venice, Italy, pp.134-139.

Sarno, R., Dewandono, R.D., Tohari, A., Mohammad Farid, M., and Sinaga, F. (2015)' Hybrid Association Rule Learning and Process Mining for Fraud Detection' *IAENG International Journal of Computer Science, Vol.*42 No.2, pp.819-9224.

Schunselaar, D., Leopold, H., Verbeek, H.M.W., Aalst, W.M.P., and Reijers, H.A. (2015b)' Fournier, F., & Mendling, J. (eds.), Configuring Configurable Process Models Made Easier: An Automated Approach. *Business Process Management Workshops, International Workshop on Process Model Collections: Management and Reuse, Lecture Notes in Business Information Processing,* pp.105-117. Springer.

Schunselaar, D.M.M, Verbeek, H.M.W., Reijers, H.A., and Aalst, W.M.P. (2015a)' YAWL in the Cloud: Supporting Process Sharing and Variability' *International Workshop on Business Process Management in the Cloud, Lecture Notes in Business Information Processing,* Vol.202, pp.367-379.

Sebu, M.L, and Ciocarlie, H. (2015)' Cross-organisational Compatibility Detection with Process Mining' *Buletinul Institutului Politehnic din Iaşi Tome LXI (LXV) Fasc.* Vol.1, pp.9-28.

Sethi, T.S., and Kantardzic, M. (2017) 'On the Reliable Detection of Concept Drift from Streaming Unlabeled Data' *Expert Systems with Applications,* Vol.82. DOI: 10.1016/j.eswa.2017.04.008.

Shershakov, S.A. (2013)' DPMine: modeling and process mining tool' *Proceedings of the 9th Central & Eastern European Software Engineering Conference,* Russia, pp.2.

Shershakov, S.A. (2015)' VTMine Framework as Applied to Process Mining Modeling' *International Journal of Computer and Communication Engineering, Vol.*4 No.3, pp.166-179.

Shugurov, I., and Mitsyuk, A. (2015)' Iskra: A Tool for Process Model Repair' *The Proceedings of ISP RAS,* Vol.27 No.3.

Son, S., Nurgroho Yahya, B., Song, M., Choi, S., Hyeon, J., Lee, B., Jang, Y., and Sung, N. (2014). Process Mining for Manufacturing Process Analysis: A case Study. *Asia Pacific conference on Business Process Management,* Brisbane, Australia.

Swapnali, B.S., and Ravi, P.P. (2015)' Process Mining by using Event Logs' *International Journal of Emerging Trend in Engineering and Basic Sciences,* Vol.2 No.1, pp.661-664, *International Journal of Computer Applications,* Vol.116 No.19, pp.31-35.

Tax N., Sidorova N., and van der Aalst W.M.P. (2017) 'Discovering More Precise Process Models from Event Logs by Filtering Out Chaotic Activities', [Accepted paper]. arXiv:1711.01287 [cs.DB].

Tax, N., Sidorova, N., Haakma, R. and van der Aalst, W.M.P. (2016) 'Event abstraction for process mining using supervised learning techniques' *the proceedings of the SAI Intelligent Systems Conference.*

Taylor, P., Leida, M., and Majeed, B. (2012). Case Study in Process Mining in a Multinational Enterprise. *Proceeding of the 1st International Symposium on Data-Driven Process Discovery and Analysis, LNBIP*, 116:134-153, Aberer, K., Damiani, E., & Dillon, T. (eds), Springer, Berlin, Heidelberg.

Van der Aalst W.M.P, de Medeiros A.K.A, Weijters A.J.M.M. (2005) 'Genetic process mining', in *Lecture Notes in Computer Science* 3536, Springer, Miami, pp. 48-69.

Van der Aalst W.M.P, Weijters A.J.M.M., and Maruster L. (2004) 'Workflow Mining: Discovering Process Models from Event Logs', *IEEE Transaction on Knowledge and Data Engineering*, Vol. 16 No.9, pp.1128–1142.

Van der Aalst W.M.P. (1997) 'Verification of Workflow Nets', in *Lecture Notes in Computer Science 1248*, Springer, Berlin, Heidelberg, pp. 407-426.

Van der Aalst W.M.P. (1998) 'The Application of Petri Nets to Workflow Management', *Journal of Circuits, Systems and Computers*, Vol. 8 No.1, pp. 21–66.

Van der Aalst W.M.P. (2011) *Process Mining: Discovery, Conformance and Enhancement of Business Processes,* 1st ed, Springer Verlag, Berlin.

van der Aalst W.M.P., de Medeiros A.K.A., Weijters A.J.M.M. (2006) Process Equivalence: Comparing Two Process Models Based on Observed Behavior. In: Dustdar S., Fiadeiro J.L., Sheth A.P. (eds) Business Process Management. BPM 2006. Lecture Notes in Computer Science, vol 4102. Springer, Berlin, Heidelberg

Van der Aalst W.M.P., Reijers H.A, Weijters A.J.M.M., van Dongen B.F, de Medeiros A.K.A, Song M., and Verbeek H.M.W. (2007) 'Business process mining: An industrial application', *Information Systems*, Vol. 32 No.5, pp. 713–732.

Van der Aalst, W. M. P. (1998). The Application of Petri Nets to Workflow Management. *Journal of Circuits, Systems and Computers*, 8:21-66.

Van der Aalst, W. M. P. (2011). *Book on Process Mining: Discovery, Conformance and Enhancement of Business Processes,* 1st ed. Springer, Berlin, Heidelberg.

Van der Aalst, W. M. P., Reijersa, H. A., Weijtersa, A. J. M. M., van Dongena, B. F., Alves de Medeirosa, A. K., Songa, M., and Verbeeka, H. M. W. (2007). Business process mining: An industrial application. *Information Systems,* 32:713-732.

Van der Aalst, W. M. P., Alves de Medeiros, A. K., and Weijters, A. J. M. M. (2005). Genetic process mining. *Proceeding of the 26th International Conference on Applications and Theory of Petri nets,* LNCS, 3536:48-69, Springer, Miami.

Van der Aalst, W. M. P., Weijters, A. J. M. M., and Maruster, L. (2003). Workflow Mining: Discovering process models from event logs. *IEEE Transaction on Knowledge Data Engineering,* 16:1128-1142.

Van der Aalst, W.M.P. (2011a) *Process Mining: Discovery, Conformance and Enhancement of Business Processes*, 1st ed., Springer-Verlag Berlin Heidelberg.

Van der Aalst, W.M.P. (2011b)' On the Representational Bias in Process Mining' *20th IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE),* Paris, pp.2-7.

Van der Aalst, W.M.P. (2013a)' Decomposing Petri Nets for Process Mining: A Generic Approach' *Distributed and Parallel Databases,* Vol.31 No.4, pp.471-507.

Van der Aalst, W.M.P. (2013b)' A General Divide and Conquer Approach for Process Minin' *Ganzha, M., Maciaszek, L., & Paprzycki, M. (eds.), IEEE Computer Society, Federated Conference on Computer Science and Information Systems,* pp.1-10.

Van der Aalst, W.M.P. (2013c)' Process Cubes: Slicing, Dicing, Rolling Up and Drilling Down Event Data for Process Mining' Song, M., Wynn, M., & Liu, J. (eds.), *Asia Pacific Conference on Business Process Management, Lecture Notes in Business*

*Information Processing, Vol.3*159, pp.1-22. Springer.

Van der Aalst, W.M.P. (2014a)' Extracting Event Data from Databases to Unleash Process Mining' *Business Process Management Roundtable 2014, 2015BPM - Driving Innovation in a Digital World,* pp.105-128, Springer.

Van der Aalst, W.M.P., and Verbeek, H.M.W. (2014b)' Process Discovery and Conformance Checking Using Passages' *Fundamenta Informaticae, Vol.*131 No.1, pp.103-138.

Van der Alast, W. M. P. *et al.* (2012). Process Mining Manifesto. *Proceeding of Business Process Management Workshops, LNBIP,* 99:169-194, Daniel, F., Barkaoui, K., & Dustdar, S. (eds), Springer, Berlin, Heidelberg.

Van Dongen B. F. *et al*. (2005). The ProM framework: A new era in process mining tool support. *Proceeding of 26th 10th International Conference on Applications and Theory of Petri Nets,* 444–454, Springer, Miami.

Van Dongen B.F, de Medeiros A.K.A, Verbeek H.M.W, Weijters A.J.M.M., and van der Aalst W.M.P. (2005) 'The ProM Framework: A New Era in Process Mining Tool Support', in *Lecture Notes in Computer Science 3536,* Ciardo G, Darondeau P (eds.), Springer, Berlin, Heidelberg, pp. 444-454.

Van Dongen, B. F., and Shabani, S. (2015). Relational XES: Data management for process mining. *Proceeding of the 27th International Conference on Advanced Information Systems Engineering,* 1367:169-176, Stockholm, Sweden.

Van Eck, M.L., Buijs, J.C.A.M., and Dongen, B.F. (2014)' Genetic Process Mining: Alignment-based Process Model Mutation' *Business Process Intelligence workshop,* Haifa, Israel.

Van Zelst, S.J., Burattin, A., Dongen, B.F., and Verbeek, H.M.W. (2014)' Data Streams in ProM 6: A Single-Node Architecture' *BPM Demos,* Haifa, Israel.

Van Zelst, S.J., Dongen, B.F., and Aalst, W.M.P. (2015)' Avoiding Over-Fitting in ILP-Based Process Discovery. *13th International Conference, BPM,* Innsbruck, Austria, Vol.9253, pp.163-171.

Vazquez Barreiros, B., Lama, M., Mucientes, M., and Vidal, J. (2014). Softlearn: A process mining platform for the discovery of learning paths. *Proceeding of the 14th*

International Conference on Learning Technologies, 373-375, Athens, Greece.

Vázquez-Barreiros, B., Chapela, D., Mucientes, M., Lama, M., and Berea, D. (2016). Process Mining in IT Service Management: A Case Study. Proceeding of the *International. Workshop on Algorithms & Theories for the Analysis of Event Data*, Toruń, Poland.

Verbeek H.M.W. and van der Aalst W.M.P. (2016b) 'Merging Alignments for Decomposed Replay' *Kordon F. & Moldt D. (eds) Application and Theory of Petri Nets and Concurrency, PETRI NETS, Lecture Notes in Computer Science,* Vol. 9698. Springer, Cham.

Verbeek, H. M. W. (2017) 'Decomposed replay using hiding and reduction as abstraction' *LNCS Transactions on Petri Nets and Other Models of Concurrency* (ToPNoC).

Verbeek, H. M. W., van der Aalst, W. M. P. and Munoz-Gama, J. (2016a) 'Divide and conquer' *BPMCenter.org, BPM Center Report BPM-16-06*.

Verbeek, H.M.W. (2014)' Decomposed Process Mining with DivideAndConquer' *BPM Demos 2014,* Haifa, Israel.

Verbeek, H.M.W., and Aalst, W.M.P. (2015)' Decomposed Process Mining: The ILP Case' *Fournier, F., Mendling, J. (eds.), BPM 2014 Workshops,* Vol.20, pp.264-276, Springer.

Vogelgesang, T., and Appelrath, H.J. (2015)' Multidimensional Process Mining with PMCube Explore' *Proceedings of the BPM Demo Session 2015 Co-located with the 13th International Conference on Business Process Management,* Innsbruck, Austria, pp.90-94.

Wang, J., Song, S., Lin, X., Zhu, X. and Pei, J. (2015) 'Cleaning structured event logs: A graph repair approach' *Proceeding of the 31st International Conference on Data Engineering*, pp.30–41.

Wang, J., Tan, S., and Wen, L. (2012)' An Empirical Evaluation of Process Mining Algorithms based on Structural and Behavioral Similarities' *Proceedings of the 27th Annual ACM Symposium on Applied Computing*, pp.211-213.

Wang, J., Wong, R.K., Ding. J., Guo, Q., and Wen, L. (2013)' Efficient selectin of Process Mining Algorithms' *IEEE transactions on services computing*, Vol.6 No.4.

Weijters A.J.M.M., van der Aalst W.M.P., de Medeiros A.K.A. (2006) 'Process Mining with the Heuristics Miner-algorithm', *BETA Working Paper Series, WP 166,* Eindhoven University of Technology, Eindhoven.

Weijters, A. J. M. M., van der Aalst, W. M. P., and Alves de Medeiros, A. K. (2006). Process

Mining with the Heuristics Miner-algorithm. *BETA Working Paper Series WP 166,* Eindhoven University of Technology, Eindhoven.

Wen L, van der Aalst W M P, Wang J, Sun J. (2007) 'Mining process models with non-free-choice constructs', *Data Mining and Knowledge Discovery*, Vol.15 No.2., pp. 145-180.

Wen L, Wang J, Sun J. (2006) 'Detecting Implicit Dependencies Between Tasks from Event Logs', in *Lecture Notes in Computer Science 3841*, Zhou X, Li J, Shen H T, Kitsuregawa M, Zhang Y(eds.), Springer, Berlin, Heidelberg, pp. 591-603.

Wen L, Wang J, Sun J. (2007) 'Mining Invisible Tasks from Event Logs', in *Lecture Notes in Computer Science 4505*, Springer, Berlin, Heidelberg, pp. 358-365.

Wen L, Wang J, van der Aalst, W M P, Huang B, Sun J. (2010) 'Mining process models with prime invisible tasks', *Data & Knowledge Engineering,* Vol.69 No.10, pp. 999- 1021.

Werf, J.M.E.M. v. d., Verbeek, H.M.W., and Aalst, W.M.P. (2012)' Context-Aware Compliance Checking' *BPM 2012 Proceedings,* Vol.7481, pp.98-113, Springer.

Werf, J.M.E.M.v.d., Dongen, B.F.v., Hurkens, C.A.J., and Serebrenik, A. (2009) 'Process Discovery using Integer Linear Programming' Fundam. Inform. Vol.94 No.3-4,pp. 387–412

Yilmaz, O. and Karagoz, P. (2015) 'Generating Performance Improvement Suggestions by using Cross-Organizational Process Mining' *SIMPDA, CEUR Workshop Proceedings*, Vol.1527, pp. 3-17.

Zaouali, S., and Ghannouchi, S. A. (2016). Proposition of an approach based on BPM to manage agile development processes. *Proceeding of the 3rd International Conference on Systems of Collaboration*, 1-6, Casablanca, Morocco.

Zellner, G. (2011). A structured evaluation of business process improvement approaches. *Journal of Business Process Management,* 17:203-237.

Zeng, Q., Sherry, X., Sun, B., Duan, H., Liu, C. and Wang, H. (2013) 'Cross-organizational collaborative workflow mining from a multi-source log' *Decision Support Systems,* Vol.54, pp.1280-1301.

Zhao, W., Liu, X., and Dai, W. (2014)' Simplified Process Model Discovery Based on Role-Oriented Genetic Mining' *The Scientific World Journal,* Vol.2014, Article ID 298592, pp.8, Hindawi.